# NINTENDO ENTERTAINMENT SYSTEM

# PROGRAMMER'S REFERENCE GUIDE

Reverse Engineered By Arti Haroutunian

NOVEMBER 1988
(Revised 7/7/89)

## TABLE OF CONTENTS

## INTRODUCTION

This document describes programming of the NINTENDO ENTERTAINMENT SYSTEM (NES) for game programmers. Thorough knowledge of the 6502 CPU and general principles of graphics and sound programming are prerequisites to understanding this document.

The entire contents of this document were discovered by the author through reverse engineering. The bulk of this work was done by disassembling and commenting the contents of several NES cartridges. Schematics of the hardware were drawn and with the aid of an oscilloscope and a logic analyzer the remaining functional facts were arrived at. Certain operating characteristics were then tested and verified through custom programs and alteration of existing programs.

This document was prepared entirely by the author with some feedback from field testers utilizing this information. The author has had no access to documentation or consultation from employees or affiliates of NINTENDO of America or their Japanese counterpart. Because of the speculative nature of the information contained herein, no guarantees are given nor implied about the accuracy of this document. The author has, however, made a reasonable effort to verify all facts in this document through test programs.

## THE HARDWARE

The NES is a dedicated game computer. It has a dual bus architecture. The Central Processing Unit (CPU) is a custom chip emulating a 6502 and controls the first bus. The Picture Processing Unit (PPU) is a custom graphics controller not emulating any graphics chip known to me. The PPU controls the second bus. Communications between the two processors is controlled by the CPU which can read and write to the PPU.

The CPU is 100% software compatible with the basic 6502. It does not support any extensions of the instructions set, such as found in the 65C02. On the hardware side it is not pin compatible and doesn't use all the signals the original 6502 does. The CPU also includes a multi channel wave synthesizer for music and sound effects. It is logically mapped into the CPU's address space at $4000 - $5FFF. The CPU includes dedicated I/O pins for game controller interfacing. 2K bytes of static RAM are mapped to $0000 - $07FF. The RAM is used for zero page storage, the stack and general variable usage. The PPU interface is mapped to $2000 - $3FFF. The CPU communicates with the PPU through this address range which contains several dedicated PPU registers. Because of the dual bus nature of the system no cycles are stolen from the CPU during the visible portion of the display cycle, resulting in true 1.79 MHz operation.

The rest of the memory space is mapped to the cartridge. Area $6000 - $7FFF is reserved for optional CPU RAM that a cartridge may contain. Area $8000 - $FFFF is reserved for the programs to be executed. Since all programs are contained in ROM, writes to this area are used to perform cartridge specific tasks such as bank switching.

The PPU is a graphics display chip handling the physical displaying of information. It gets instructions from the CPU during a brief interval in the NMI service routine and operates autonomously thereafter refreshing the display. The motherboard contains 2K bytes of static RAM used as screen memory. The screen memory area is mapped to $2000 - $2FFF, allowing a total of 4 separate screens. Because the existing 2K only allows 2 full screens some of the screen address selection logic is handled on the cartridge which decides on horizontal or vertical scrolling and may even add an extra 2K for true diagonal scrolling. Area $0000 - $1FFF on the cartridge may contain RAM or ROM for background and sprite definitions. This area may contain bank switched memory which is switched by the CPU.

The PPU generates character based screens and supports sprites. There are no signals generated for the CPU to allow for line interrupts, the only signal generated is a vblank signal that is used as an NMI on the CPU. All dynamic display changes must be handled through exact cycle counting with the CPU. Sprites are defined in vblank and also cannot be altered dynamically during the visible cycle of a screen refresh. The PPU is clocked by a master clock at 21 Mhz which generates all video clocking signals and feeds an R/F modulator.

The motherboard has no provisions for on board expansion of memory. It does have a separate edge connector on the bottom giving access to many but not all signals. I have never encountered a product using this expansion connector.

A security chip fed by a 4 MHz clock generates a signature that communicates with cartridges. Unless a cartridge responds with the proper signature in return the security chip enables the reset lines to both CPU and PPU and resends the signature periodically. This causes a blank screen to be displayed when no cartridge is inserted and also ensures that every cartridge must contain an NES proprietary security chip. The motherboard contains no ROM.

The entire unit is housed in a plastic box with insufficient ventilation. A cartridge running several hours generates a lot of damaging heat. One very effective solution to this is removing the cartridge door from the NES console. This results in increased airflow and a significant reduction in ambient cartridge temperature.

## GRAPHICS PROGRAMMING

The NES has a character based screen and includes sprite support. Memory areas for the PPU include screen memory on the motherboard and background and sprite character definition memory on the cartridge. The CPU communicates with the PPU through several PPU registers mapped into the CPU's address space. The CPU can access PPU memory only through communicating with the PPU, not directly because of separate buses.

## SCREEN MEMORY

A full NES screen consists of 32 characters across by 30 characters high. Each character is eight pixels square yielding a total resolution of 256 x 240 pixels. This uses up a total of 960 bytes per character screen. The NES doesn't use a blank overscan area, the characters fill the entire physical viewing area of the screen. Because of this you might lose some display information depending on the particular display model used. I have displayed worst case display loss on a Commodore 1702 video monitor. You lose about 1 1/2 lines form the top, 1 line from the bottom and parts of the left and right borders due to screen curvature. Losses are less severe on regular television screens which most NES game players use.

Screen memory is followed by 64 bytes of background color memory for a total of 1K Bytes per screen. Therefore internal memory allows a total of two full screens to be stored, and the screens are mapped as follows:

```
$2000 - $23BF  Screen 1 character map
$23C0 - $23FF  Screen 1 color memory
$2400 - $27BF  Screen 2 character map
$27C0 - $27FF  Screen 2 color memory
```

The CPU can write data to the screen by using the PPU Memory Address Register (PMAR, $2006) and the PPU Memory Data Register (PMDR, $2007). Data is transferred by writing a base destination address to PMAR followed by byte by byte data writes to PMDR. The destination address can be programmed to either autoincrement by one (for row transfers) or autoincrement by 32 (for column transfers). Autoincrement mode is set through PPU Control Register 1 (PCR1, $2000), bit 2. Since VCR1 seems to be a write only registers it is a good idea to maintain a shadow register in RAM. VCR1 contains several functions and a shadow register allows alteration of a single function without disturbing the rest.

Screen updates are accomplished by writing the 16 bit PPU destination address to PMAR (MSB first) followed by the data bytes.

For example, to write a column of data to the third column in the display you may use the following code:

```
PCR1            EQU   $2000       ; Video Control Reg 1
ShPCR1          EQU   $300        ; RAM shadow
PMAR            EQU   $2006       ; PPU Mem Adr Reg
PMDR            EQU   $2007       ; PPU Mem Data Reg

AIRMask         EQU   %11111011 ; auto inc bit mask
AIR1            EQU   %00000000 ; inc by 1
AIR32           EQU   %00000100 ; inc by 32

SCREEN1BASE     EQU   $2000       ; start of screen 1

Ind             EQU   $F0         ; pointer to src data

WriteColumn     LDA   #>SCREEN1BASE
                STA   PMAR        ; store MSB of dest
                LDA   #<SCREEN1Base
                CLC
                ADC   #2          ; add offset
                STA   PMAR        ; store LSB of dest
                LDA   ShVCR1      ; get shadow reg
                AND   #AIRMask    ; isolate bit
                OR    #AIR32      ; set 32 byte mode
                STA   ShPCR1      ; save shadow
                STA   PCR1        ; set register
                LDY   #0          ; clear index
Loop            LDA   (Ind),Y     ; get source data
                STA   PMDR        ; save to dest
                INY
                CPY   #32
                BCC   Loop
                RTS
```

To read screen data the reverse of the above procedure could be used. Note, however, that the very first byte you read is invalid and should be ignored.

For example, to read the third column of data from the display you may use the following code:

```
PCR1            EQU   $2000       ; Video Control Reg 1
ShPCR1          EQU   $300        ; RAM shadow
PMAR            EQU   $2006       ; PPU Mem Adr Reg
PMDR            EQU   $2007       ; PPU Mem Data Reg

AIRMask         EQU   %11111011 ; auto inc bit mask
AIR1            EQU   %00000000 ; inc by 1
AIR32           EQU   %00000100 ; inc by 32

SCREEN1BASE     EQU   $2000       ; start of screen 1
```

```
Ind             EQU  $F0        ; pointer to src data

ReadColumn      LDA  #>SCREEN1BASE
                STA  PMAR       ; store MSB of dest
                LDA  #<SCREEN1Base
                CLC
                ADC  #2         ; add offset
                STA  PMAR       ; store LSB of dest
                LDA  ShVCR1     ; get shadow reg
                AND  #AIRMask   ; isolate bit
                OR   #AIR32     ; set 32 byte mode
                STA  ShPCR1     ; save shadow
                STA  PCR1       ; set register
                LDY  #0         ; clear index
                LDA  PMDR       ; discard first byte
Loop            LDA  PMDR       ; get screen data
                STA  (Ind),Y    ; save to dest
                INY
                CPY  #32
                BCC  Loop
                RTS
```

## SCROLLING

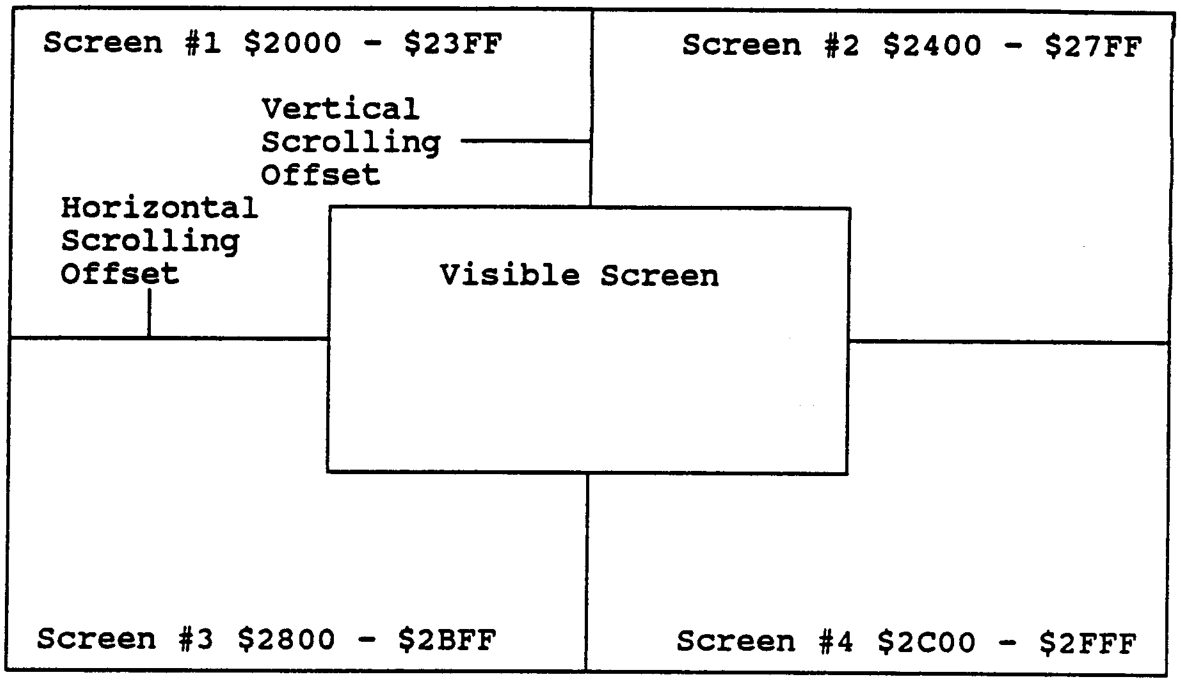
The screen may be fine scrolled both horizontally and vertically. Scrolling is controlled through the PPU Scroll Offset Register (PSOR, $2005) and PCR1. Each scroll value may be from 0 to 256. A value of 256 is specified by setting the respective upper scroll bit in PCR1. Values from 0 to 255 are transferred to PSOR through two consecutive writes, with the horizontal value written first.

For example, to transfer scroll values to the screen you may use the following code:

```
PCR1            EQU  $2000      ; PPU Ctrl Reg 1
PSOR            EQU  $2005      ; PPU Scroll Reg

HScroll         EQU  $300       ; horiz offset
HScrollH        EQU  $301       ; horiz hi bit
VScroll         EQU  $302       ; vert offset
VScrollH        EQU  $303       ; vert hi bit

DoScroll        LDA  HScrollH   ; see if H at max
                LSR  A
                BCC  DS1        ; not at max
                LDA  ShPCR1     ; get shadow
                ORA  #1         ; set H hi bit
                STA  ShPCR1     ; save
```

```
DS1             LDA  VScrollH   ; see if V at max
                LSR  A
                BCC  DS2        ; not at max
                LDA  ShPCR1     ; get shadow
                ORA  #2         ; set V hi bit
                STA  ShPCR1     ; save shadow
DS2             LDA  ShPCR1     ; get shadow
                STA  PCR1       ; into register
                LDA  HScroll    ; get h value
                STA  PSOR       ; into reg
                LDA  VScroll    ; get v value
                STA  PSOR       ; into reg
                RTS
```

The NES has 4 logical screens. They are related to the visible area as follows:



Since the NES has only 2K bytes of screen memory we have available 2 physical screens. On most cartridge types you are given the option of how to assign physical screens to logical screens for scrolling purposes. This is achieved through either a hardware Horizontal/Vertical jumper on the cartridge or a special purpose chip which can be programmed for H/V selection.

Physical to logical screen mapping is achieved through cartridge logic generating PPU address 11 for screen memory.

When horizontal scrolling is selected the following mapping is in effect. Physical screen #1 is mapped to logical screens #1 and #3 simultaneously. Physical screen #2 is mapped to logical screens #2 and #4 simultaneously. Applying a horizontal scrolling offset will result in true scrolling from one physical screen to another. However, applying a vertical offset will result in the same physical screen wrapping around the edges. To achieve diagonal scrolling the vertical scrolling data must be dynamically rewritten.

When vertical scrolling is selected the following mapping is in effect. Physical screen #1 is mapped to logical screens #1 and #2 simultaneously. Physical screen #2 is mapped to logical screens #3 and #4 simultaneously. Applying a vertical scrolling offset will result in true scrolling from one physical screen to another. However applying a horizontal offset will result in the same physical screen wrapping around the edges. To achieve diagonal scrolling the horizontal scrolling data must be dynamically rewritten.

Some cartridges allow for character definition memory to be used as extra screen memory, resulting in 4 physical screens and an exact one to one mapping of logical to physical screens. This is a vehicle for true diagonal scrolling with a minimum of software overhead.

## CHARACTER SET

Background characters consist of an 8 x 8 pixel array. Each pixel can be one of four colors, requiring 2 bits per pixel. Therefore a total of 128 bits or 16 bytes are required to define a character. Characters are defined in a contiguous block of 16 bytes. The definition is structured similar to IBM EGA graphics. The character has two bit planes, the first eight bytes define the first bit plane and the second eight bytes define the second bit plane. Consider the following character, with each number defining a pixel's two bit color value (0 - 3):

```
0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 2 0 0 0 0 0
0 0 0 3 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 2 0 0
0 0 0 0 0 0 3 0
0 1 2 3 0 1 2 3
```

A data statement describing this character would look like the following (using binary constants):

```
        ; first plane
DB      %00000000,%01000000,%00000000,%00010000
DB      %00001000,%00000000,%00000010,%01010101

        ; second plane
DB      %00000000,%00000000,%00100000,%00010000
DB      %00000000,%00000100,%00000010,%00110011
```

The background character set consists of 256 characters requiring a total of 4K bytes of memory. This memory is on the cartridge, and may be either RAM or ROM. The background character set may be mapped starting at $0000 or $1000. Mapping is controlled by PCR1, bit 4. When bit 4 = 0, the character set is mapped to $0000 - $0FFF, otherwise the set is mapped to $1000 - $1FFF.

Even though character mapping is restricted to two areas, several cartridges offer bank switching mechanisms for the background character set memory. This allows you access to several different background character sets.

A game using the above technique on a single screen is 'Mickey Mouscapades'. The background character set is switched in the middle of the title screen to allow for a screen made up of more than 256 individual characters.

## SCREEN COLORS

The NES has four color sets made up of four colors each for use by the background characters. Each color is described by a byte where

```
Bits 3..0 - primary color
Bits 5..4 - intensity
Bits 7..6 - not used
```

The primary colors seem to be:

| | |
|---|---|
| 0 | Gray |
| 1 | Blue 1 |
| 2 | Blue 2 |
| 3 | Blue 3 |
| 4 | Purple |
| 5 | Red/Pink |
| 6 | Red |
| 7 | Red/Orange |
| 8 | Brown/Yellow |
| 9 | Green 1 |
| A | Green 2 |
| B | Green/Turquoise |

| | |
|---|---|
| C | Cyan |
| D | Gray 2 |
| E | Black |
| F | Black |

The color sets are mapped as follows:

| | |
|---|---|
| $3F00 | color set #1 background (not used) |
| $3F01 | color set #1 color for bit combo 01 |
| $3F02 | color set #1 color for bit combo 10 |
| $3F03 | color set #1 color for bit combo 11 |
| $3F04 | color set #2 background (not used) |
| $3F05 | color set #2 color for bit combo 01 |
| $3F06 | color set #2 color for bit combo 10 |
| $3F07 | color set #2 color for bit combo 11 |
| $3F08 | color set #3 background (not used) |
| $3F09 | color set #3 color for bit combo 01 |
| $3F0A | color set #3 color for bit combo 10 |
| $3F0B | color set #3 color for bit combo 11 |
| $3F0C | color set #4 background (not used) |
| $3F0D | color set #4 color for bit combo 01 |
| $3F0E | color set #4 color for bit combo 10 |
| $3F0F | color set #4 color for bit combo 11 |

To define a set of background character color sets the following code may be used.

```
Ind             EQU  $F0
PMAR            EQU  $2006
PMDR            EQU  $2007

SetColors       LDA  #<ColorSet
                STA  Ind          ; high byte of table
                LDA  #>ColorSet
                STA  Ind+1        ; low byte of table
                LDY  #0
                LDA  #$3F         ; high byte of dest
                STA  PMAR
                LDA  #0           ; lo byte of dest
                STA  PMAR
Loop            LDA  (Ind),Y      ; get src
                STA  PMDR         ; store in PPU
                INY
                CPY  #16
                BCC  Loop
                RTS
```

```
ColorSet        DB    1,2,3,4    ; color set #1
                DB    5,6,7,8    ; color set #2
                DB    1,2,3,5    ; color set #3
                DB    9,8,7,6    ; color set #4
```

The screen background color is specified at location $3F10.

Color sets are assigned to characters as follows: First divide the screen into blocks of 4 x 4 characters starting at the top left of the screen. You will end up with 64 blocks with one color memory location assigned to each block. Further divide each block into 4 subblocks of 2 x 2 characters. Number these 1 through 4 starting at the top left subblock. The first character block on the screen would look as follows (character positions in decimal):

```
000 001 002 003
032 033 034 035
064 065 066 067
096 097 098 099
```

Subblock 1 consists of characters 0, 1, 32 and 33.
Subblock 2 consists of characters 2, 3, 34 and 35.
Subblock 3 consists of characters 64, 65, 96 and 97.
Subblock 4 consists of characters 66, 67, 98 and 99.

Since there are four color sets two bits are required to select a color set for a subblock. Four subblocks require a byte and therefore each block is assigned a byte from color memory. The byte pairs are arranged as follows:

```
Subblock  1    bits 1,0
Subblock  2    bits 3,2
Subblock  3    bits 5,4
Subblock  4    bits 7,6
```

It follows that each subblock is restricted to three colors plus background. The respective color memory bit pair selects one of four color sets and the pixel bit pair selects the color from the chosen color set.

A color memory value of $D1 (%11010001) assigns color set 2 to subblock 1, color set 1 to subblock 2, color set 2 to subblock 3 and color set 4 to subblock 4.

These color restrictions are quite severe, especially in light of the incompatibility with existing paint programs. The four color per subblock restriction disallows straight conversions from existing multi color pictures (such as in Deluxe Paint II EGA/VGA format). Pictures should either be drawn specifically for the NES using special tools, or you may use low color resolution pictures (4 color CGA) and add colors as you go along.

## BACKGROUND GRAPHICS EXAMPLE CODE

Following is a working code segment that creates a screen out of data segments and displays it on the NES.

```
                ORG     $C000

                ; NES Equates

PCR1            EQU     $2000       ; PPU Control Register 1
PCR2            EQU     $2001       ; PPU Control Register 2
PSTR            EQU     $2002       ; PPU status register
PPCR            EQU     $2003       ; PPU DMA page count
PSOR            EQU     $2005       ; PPU Scrolling Offset Register
PMAR            EQU     $2006       ; PPU Memory Address Register
PMDR            EQU     $2007       ; PPU Memory Data Register

V1R1            EQU     $4000       ; Voice 1 Register 1
V1R2            EQU     $4001       ; Voice 1 Register 2
V1R3            EQU     $4002       ; Voice 1 Register 3
V1R4            EQU     $4003       ; Voice 1 Register 4

V2R1            EQU     $4004       ; Voice 2 Register 1
V2R2            EQU     $4005       ; Voice 2 Register 2
V2R3            EQU     $4006       ; Voice 2 Register 3
V2R4            EQU     $4007       ; Voice 2 Register 4

V3R1            EQU     $4008       ; Voice 3 Register 1
V3R2            EQU     $4009       ; Voice 3 Register 2
V3R3            EQU     $400A       ; Voice 3 Register 3
V3R4            EQU     $400B       ; Voice 3 Register 4

V4R1            EQU     $400C       ; Voice 4 Register 1
V4R2            EQU     $400D       ; Voice 4 Register 2
V4R3            EQU     $400E       ; Voice 4 Register 3
V4R4            EQU     $400F       ; Voice 4 Register 4

V5R1            EQU     $4010       ; Voice 5 Register 1
V5R1            EQU     $4011       ; Voice 5 Register 2
V5R1            EQU     $4012       ; Voice 5 Register 3
V5R1            EQU     $4013       ; Voice 5 Register 4

DPNR            EQU     $4014       ; DMA Page Number Register
VMER            EQU     $4015       ; Voice Master Enable Register


ShPCR1          EQU     $14         ; PCR1 shadow register
```

```
ShPCR2        EQU    $15        ; PCR2 shadow register
VScroll       EQU    $16        ; vertical scroll offset
HScroll       EQU    $17        ; horizontal scroll offset

PCR1Init      EQU    $10        ; clear scrolling MSB's
                                ; set auto increment to 1
                                ; sprite chars at $0000 - $0FFF
                                ; background chars at $1000 -
$1FFF
                                ; interrupt disabled
PCR2Init      EQU    $06        ; enable color
                                ; blank sprites
                                ; blank background
                                ; disable left border

              ; program variables

CodeState     EQU    $0A        ; software state
Temp          EQU    $F6        ; temporary
Ind0          EQU    $F8        ; pointer

RESVector     SEI
              LDA    #0             ; init control registers
              STA    PCR1
              STA    PCR2
Wait0         LDA    PSTR           ; wait for hardware
              BPL    Wait0          ; to stabilize
Wait1         LDA    PSTR
              BPL    Wait1
Wait2         LDA    PSTR
              BPL    Wait2
              LDX    #0
              STX    CodeState      ; clear software state
              STX    HScroll        ; clear H scroll
              STX    VScroll        ; clear V Scroll
              STX    VMER           ; quiet all sound
              DEX
              TXS                   ; init stack pointer
              LDA    #PCR1Init      ; init PCR1
              STA    ShPCR1
              STA    PCR1
              LDA    #PCR2Init      ; init PCR2
              STA    ShPCR2
              STA    PCR2
              LDA    ShPCR1         ; enable NMI interrupts
              ORA    #$80
              STA    ShPCR1
              STA    PCR1
```

```
ForeGround     NOP                     ; do nothing
               NOP                     ; in the foreground
               NOP
               JMP     ForeGround

IRQVector      CLI                     ; should never happen
               RTI

NMIVector      PHP                     ; save processor registers
               PHA
               TXA
               PHA
               TYA
               PHA

               LDA     PSTR            ; clear interrupt bit
               LDA     ShPCR1          ; get PCR1 shadow
               AND     #$7F            ; disable interrupts
               STA     PCR1            ; save in register
               STA     ShPCR1          ; save in shadow

               LDA     #$00            ; DMA page count - 1
               STA     $2003
               LDA     #$02            ; DMA page number
               STA     DPNR

               LDA     ShPCR2          ; get PCR 2 shadow
               AND     #$E7            ; blank screen and
               STA     ShPCR2          ; and enable CPU access
               STA     PCR2

               JSR     DoGTest

               LDA     ShPCR2          ; get PCR 2 shadow
               ORA     #$18            ; enable screen and
               STA     ShPCR2          ; block CPU access
               STA     PCR2

               LDA     HScroll         ; refresh scroll registers
               STA     PSOR
               LDA     VScroll
               STA     PSOR

               LDA     ShPCR1          ; get PCR1 shadow
               ORA     #$80            ; enable interrupts
               STA     PCR1            ; save to PPU
               STA     ShPCR1          ; save to shadow
```

```
            PLA                     ; restore registers
            TAY
            PLA
            TAX
            PLA
            PLP
            RTI

            ; actual test code
            ; broken into two parts
            ; for timing considerations

DoGtest     LDA     CodeState
            CMP     #0
            BNE     GTest1
            JSR     ClrChars        ; clear screen
            JSR     ClrSprites      ; clear sprite definitions
            JSR     ClrScreen       ; clear character
definitions
            INC     CodeState       ; go to next state
            RTS

Gtest1      CMP     #1
            BNE     Gtest2
            JSR     LoadScreen      ; load actual screen and
            INC     CodeState       ; colors etc.
GTest2      RTS

            ; This routine will transfer blocks
            ; from the database to their
            ; respective destinations.
            ; Blocks include screen, color
            ; and character definitions

LoadScreen  LDA     #<ScreenData    ; get starting address
            STA     Ind0            ; of data
            LDA     #>ScreenData
            STA     Ind0+1
Lscr0       LDA     PSTR            ; clear any interrupts
            LDY     #$00
            LDA     (Ind0),Y        ; dest address low
            STA     Temp
            INY
            LDA     (Ind0),Y        ; dest address high
            STA     Temp+1
            AND     Temp            ; done if address
            CMP     #$FF            ; = $FFFF
```

```
               BEQ   LscrX
               INY
               LDA   Temp+1         ; get MSB of dest
               STA   PMAR
               LDA   Temp           ; get LSB of dest
               STA   PMAR
               LDA   #$00           ; get negative of
               SEC                  ; 16 bit data count
               SBC   (Ind0),Y
               STA   Temp
               INY
               LDA   #$00
               SBC   (Ind0),Y
               STA   Temp+1
               INY
Lscr1          LDA   (Ind0),Y       ; transfer actual data
               STA   PMDR
               INY
               BNE   Lscr2
               INC   Ind0+1         ; bump source
Lscr2          INC   Temp           ; check counter
               BNE   LScr1          ; for more data
               INC   Temp+1
               BNE   LScr1
               CLC                  ; update pointer
               TYA                  ; to next data block
               ADC   Ind0
               STA   Ind0
               BCC   Lscr4
               INC   Ind0+1
Lscr4          JMP   Lscr0
LscrX          RTS

               ; clear the screen
               ; fill with character $FF

ClrScreen      LDA   #$20           ; dest MSB = $20
               STA   PMAR
               LDA   #0             ; dest LSB = $00
               STA   PMAR
               LDX   #$00
               LDY   #$00
               LDA   #$FF
ClrScreen0     STA   PMDR           ; store character
               DEX                  ; $400 times
               BNE   ClrScreen0
               INY
               CPY   #$04
```

```
                BNE   ClrScreen0
                RTS

                ; clear the background
                ; character set area

ClrChars        LDA   #$10
                STA   PMAR
                LDA   #$00
                STA   PMAR
                LDA   #0
                TAX
                TAY
ClrCh           STA   PMDR
                INX
                BNE   ClrCh
                LDA   PSTR
                INY
                CPY   #$10
                BNE   ClrCh
                RTS

                ; clear the sprite
                ; character area

ClrSprites      LDA   #$00
                STA   PMAR
                LDA   #$00
                STA   PMAR
                LDA   #0
                TAX
                TAY
ClrSp           STA   PMDR
                INX
                BNE   ClrSp
                INY
                CPY   #$10
                BNE   ClrSp
                RTS

ScreenData      ; the first block contains the screen data
                DW    $2000          ; destination address
                DW    $0400          ; data count
                DB    0,1,2,3        ; total of 1024 screen
                                     ; bytes and color memory

                ; the second block contains the background
                ; color sets
```

```
DW     $3F00          ; destination address
DW     16             ; data count
DB     1,2,3,4        ; total of 16 color
                      ; definition bytes

; the third block contains the character
; set definition
DW     $1000          ; destination address
DW     $1000          ; data count
DB     1,2,3,4        ; total of 4096 character
                      ; definition bytes.

; terminator
DW     $FFFF

; 6502 hardware vectors

ORG    $FFFA

DW     NMIVector      ; NMI service routine
DW     RESVector      ; Reset entry routine
DW     IRQVector      ; IRQ service routine

END
```

## SPRITE PROGRAMMING

The NES has 64 physical sprites. Each sprite consists of a 8 x 8 pixel character. Characters are defined the same way as screen characters are defined. The sprite character set may be mapped to either $0000 - $0FFF (PCR1, bit 3 = 0) or $1000 - $1FFF (PCR1, bit 3 = 1).

Each sprite is defined by 4 bytes as follows:

```
Byte 0:    Sprite Y Position.
Byte 1:    Sprite Character number
Byte 2:    Sprite color, priority and orientation. Bits
           0 and 1 are used to select a color set. Bit
           5, when set, gives background characters
           priority over sprites. Bit 7, when set, flips
           the sprite horizontally. Bit 6, when set,
           flips the sprite vertically. Bits 7 and 6 may
           be used together for diagonal mirroring.
Byte3:     Sprite X Position.
```

Sprite data is generated by the CPU and placed in a special memory area ($200 - $2FF). During vblank a DMA command is used to transfer the entire sprite data page to the PPU. It therefore follows that sprite modifications during the active display cycle are not possible on the NES.

The NES limits sprite usage to eight sprites per horizontal line. Any more sprites per line get ignored by the PPU and do not show up on the screen. This limitation may only be overcome by using sprite multiplexing. It is a commonly used technique in NES games, easily recognized by the annoying sprite flicker it produces. However, it is the only way to show more than eight sprites per line. Many scrolling games use sprites to provide a stationary score/status display. Because this may use up to eight sprites, sprite multiplexing becomes inevitable. If sprite multiplexing is not desired the game should be carefully designed with the sprite limitations observed from the very beginning.

Since a total of 64 sprites are supported, and each sprite is specified by four bytes a total of 256 bytes or 1 page of memory are required. This one page is sent to the PPU during each VBlank service routine.

The DMA transfers have been observed empirically. They are activated in the VBlank code as one of the first actions taken. This code uses the PPU DMA Page Count Register (PDPC, $2003) and the DMA Page Number Register (DPNR, $4014). The code looks as follows:

```
PDPC            EQU  $2003      ; PPU DMA Page Count
DPNR            EQU  $4041      ; DMA Page Number

                LDA  #0         ; page count
                STA  PDPC
                LDA  #2         ; page number
                STA  DMAR
```

Following is a subroutine that formats sprite data into the required 4 bytes and places it into the appropriate memory for DMA transfer to the PPU. Routines like this should be located in VBlank after the current DMA has taken place. If sprite data formatting is done asynchronously with VBlank data may be transferred to the PPU at any point in time, resulting in bad or incomplete sprites displayed.

```
SpLoc           EQU  $200       ; sprite data base
SpriteYLoc      EQU  SpLoc      ; Y offset in pixels
SpriteChar      EQU  SpLoc+1    ; character number
SpriteCtl       EQU  SpLoc+2    ; sprite attributes
SpriteXLoc      EQU  SpLoc+3    ; X offset in pixels

YLoc            DB   80         ; Y = 80
XLoc            DB   100        ; X = 100
SpChar          DB   $34        ; character $34
SpCol           DB   $01        ; color set 1
SpHFlip         DB   $01        ; horizontal flip on
SpVFlip         DB   $00        ; vertical flip off
SpPrior         DB   $00        ; front of background
SpNum           DB   $05        ; physical sprite #

SetSprite       LDA  SpNum      ; get physical sprite #
                ASL  A          ; multiply by four
                ASL  A          ; for proper offset
                TAX             ; into index
                LDA  YLoc       ; transfer Y location
                STA  SpriteYLoc,X
                LDA  XLoc       ; transfer X location
                STA  SpriteXLoc,X
                LDA  SprChar    ; transfer character
                STA  SpriteChar,X
                LDA  SpCol
                STA  Temp       ; save color
                LDA  SpPrior    ; check for prior bit
                LSR  A
                BCC  SS1        ; no priority bit
                LDA  Temp
                ORA  #$20       ; set priority bit
                STA   Temp
SS1             LDA   SpHFlip     ; check for H flip
                LSR   A
                BCC   SS2         ; no H flip
                LDA   Temp
                ORA   #$40        ; set H flip bit
                STA   Temp
SS2             LDA   SpVFlip     ; check for V flip
                LSR   A
                BCC   SS3         ; no V flip
                LDA   Temp
                ORA   #$80        ; set V flip bit
                STA   Temp
SS3             LDA   Temp        ; get attributes
                STA   SpriteCtl,X
                RTS
```

Sprites are prioritized amongst themselves by their position index. The lower their index, the higher their priority. This applies to the eight sprite limit. If you try to display sprites 1 through 10 on the same line sprites 9 and 10 will be blanked by the hardware. There is no hardware detection mechanism accessible by the program for detecting the eight sprite limitation.

The most common way of disabling sprites is setting the Y coordinate to a value of $F8. This seems to move the sprite off the bottom of the screen and effectively disables that sprite.

Sprite to background priority can be controlled through bit 4 of SPRITECTL. If bit 4 = 0, sprites appear above the background, otherwise the sprites will be behind background graphics.

Since each sprite has its own color set selection the color limitations existing for background characters do not apply to sprites.

## SPRITE COLORS

Sprite colors are arranged similarly to character colors. Following is a map of the color locations in PPU memory:

| | |
|---|---|
| $3F11 | color set #1 color for bit combo 01 |
| $3F12 | color set #1 color for bit combo 10 |
| $3F13 | color set #1 color for bit combo 11 |
| $3F14 | color set #2 background (not used) |
| $3F15 | color set #2 color for bit combo 01 |
| $3F16 | color set #2 color for bit combo 10 |
| $3F17 | color set #2 color for bit combo 11 |
| $3F18 | color set #3 background (not used) |
| $3F19 | color set #3 color for bit combo 01 |
| $3F1A | color set #3 color for bit combo 10 |
| $3F1B | color set #3 color for bit combo 11 |
| $3F1C | color set #4 background (not used) |
| $3F1D | color set #4 color for bit combo 01 |
| $3F1E | color set #4 color for bit combo 10 |
| $3F1F | color set #4 color for bit combo 11 |

## SPRITE EXAMPLE CODE

Following is a working code segment that displays and animates sprites from data statements.

```
                ORG     $8000

FrameData       ; contains the frame data describing
                ; what components a sprite is made of
                ; there are 128 definitions each using
                ; 108 bytes.
                DB      1,2,3,4   ; $3600 frame bytes

                ORG     $BF00

AnimData        ; contains the current animation data
                DB      1,2,3,4

                ORG     $C000

                ; NES Equates

PCR1            EQU     $2000     ; PPU Control Register 1
PCR2            EQU     $2001     ; PPU Control Register 2
PSTR            EQU     $2002     ; PPU status register
PPCR            EQU     $2003     ; PPU DMA page count
PSOR            EQU     $2005     ; PPU Scrolling Offset Register
PMAR            EQU     $2006     ; PPU Memory Address Register
PMDR            EQU     $2007     ; PPU Memory Data Register

V1R1            EQU     $4000     ; Voice 1 Register 1
V1R2            EQU     $4001     ; Voice 1 Register 2
V1R3            EQU     $4002     ; Voice 1 Register 3
V1R4            EQU     $4003     ; Voice 1 Register 4

V2R1            EQU     $4004     ; Voice 2 Register 1
V2R2            EQU     $4005     ; Voice 2 Register 2
V2R3            EQU     $4006     ; Voice 2 Register 3
V2R4            EQU     $4007     ; Voice 2 Register 4

V3R1            EQU     $4008     ; Voice 3 Register 1
V3R2            EQU     $4009     ; Voice 3 Register 2
V3R3            EQU     $400A     ; Voice 3 Register 3
V3R4            EQU     $400B     ; Voice 3 Register 4

V4R1            EQU     $400C     ; Voice 4 Register 1
V4R2            EQU     $400D     ; Voice 4 Register 2
V4R3            EQU     $400E     ; Voice 4 Register 3
```

```
V4R4          EQU   $400F     ; Voice 4 Register 4

V5R1          EQU   $4010     ; Voice 5 Register 1
V5R1          EQU   $4011     ; Voice 5 Register 2
V5R1          EQU   $4012     ; Voice 5 Register 3
V5R1          EQU   $4013     ; Voice 5 Register 4

DPNR          EQU   $4014     ; DMA Page Number Register
VMER          EQU   $4015     ; Voice Master Enable Register


ShPCR1        EQU   $14       ; PCR1 shadow register
ShPCR2        EQU   $15       ; PCR2 shadow register
VScroll       EQU   $16       ; vertical scroll offset
HScroll       EQU   $17       ; horizontal scroll offset

PCR1Init      EQU   $10       ; clear scrolling MSB's
                              ; set auto increment to 1
                              ; sprite chars at $0000 - $0FFF
                              ; background chars at $1000 -
$1FFF
                              ; interrupt disabled
PCR2Init      EQU   $06       ; enable color
                              ; blank sprites
                              ; blank background
                              ; disable left border

              ; program variables

CodeState     EQU   $0A       ; software state
Temp          EQU   $F6       ; temporary
Ind0          EQU   $F8       ; pointer



RESVector     SEI
              LDA   #0                ; init control registers
              STA   PCR1
              STA   PCR2
Wait0         LDA   PSTR              ; wait for hardware
              BPL   Wait0             ; to stabilize
Wait1         LDA   PSTR
              BPL   Wait1
Wait2         LDA   PSTR
              BPL   Wait2
              LDX   #0
              STX   CodeState         ; clear software state
              STX   HScroll           ; clear H scroll
```

```
                STX    VScroll         ; clear V Scroll
                STX    VMER            ; quiet all sound
                DEX
                TXS                    ; init stack pointer
                LDA    #PCR1Init       ; init PCR1
                STA    ShPCR1
                STA    PCR1
                LDA    #PCR2Init       ; init PCR2
                STA    ShPCR2
                STA    PCR2
                LDA    ShPCR1          ; enable NMI interrupts
                ORA    #$80
                STA    ShPCR1
                STA    PCR1

ForeGround      NOP                    ; do nothing
                NOP                    ; in the foreground
                NOP
                JMP    ForeGround

IRQVector       CLI                    ; should never happen
                RTI

NMIVector       PHP                    ; save processor registers
                PHA
                TXA
                PHA
                TYA
                PHA

                LDA    PSTR            ; clear interrupt bit
                LDA    ShPCR1          ; get PCR1 shadow
                AND    #$7F            ; disable interrupts
                STA    PCR1            ; save in register
                STA    ShPCR1          ; save in shadow

                LDA    #$00            ; DMA page count - 1
                STA    $2003
                LDA    #$02            ; DMA page number
                STA    DPNR

                LDA    ShPCR2          ; get PCR 2 shadow
                AND    #$E7            ; blank screen and
                STA    ShPCR2          ; enable CPU access
                STA    PCR2

                JSR    DoSTest
                LDA   ShPCR2        ; get PCR 2 shadow
                ORA   #$18          ; enable screen and
                STA   ShPCR2        ; block CPU access
                STA   PCR2

                LDA   HScroll       ; refresh scroll registers
                STA   PSOR
                LDA   VScroll
                STA   PSOR

                LDA   ShPCR1        ; get PCR1 shadow
                ORA   #$80          ; enable interrupts
                STA   PCR1          ; save to PPU
                STA   ShPCR1        ; save to shadow

                PLA                 ; restore registers
                TAY
                PLA
                TAX
                PLA
                PLP
                RTI

                ; actual sprite display code

DoStest         LDA   CodeState
                CMP   #0
                BNE   STest1
                JSR   ClrChars
                JSR   ClrSprites
                JSR   ClrScreen
                INC   CodeState
                RTS

Stest1          CMP   #$01
                BNE   Stest2
                JSR   LoadSprites
                INC   CodeState
                RTS

Stest2          CMP   #2
                BNE   Stest3
                JSR   InitSprites
                INC   CodeState
                RTS

STest3          CMP   #3
                BNE   Stest4
```

```
               JSR   DoSprites
STest4         RTS

               ; Sprite working variables

SprPtr         EQU   $20         ; two byte ptr to beginning of
sprite def
SprColIdx      EQU   $22         ; current sprite color offset
SprDatIdx      EQU   $23         ; current sprite data offset
SprChrIdx      EQU   $24         ; physical sprite character
SprRow         EQU   $25         ; sprite row
SprColumn      EQU   $26         ; sprite column
SprRowOfs      EQU   $27         ; row offset in pixels
SprColOfs      EQU   $28         ; column offset in pixels

StepPtr        EQU   $100        ; points to current animation
step
CurrX          EQU   $101        ; current X coordinate
CurrY          EQU   $102        ; current Y coordinate
StepIndex      EQU   $103        ; index of current step
StepFrame      EQU   $104        ; current frame used
StepDelay      EQU   $105        ; delay for this frame
StepRep1       EQU   $106        ; repeat count 1
StepRep2       EQU   $107        ; repeat count 2

               ; initialize the sprite parameters

InitSprites    LDA   #0
               STA   StepPtr         ; clear step ptr
               LDA   #100
               STA   CurrX           ; X = 100
               STA   CurrY           ; Y = 100
               LDY   #$06            ; switch bank 6 into
               JSR   SetBank         ; area $8000 - $BFFF
               LDA   #0
               STA   StepIndex       ; step index
               STA   StepDelay       ; step delay
               STA   SprChrIdx       ; character index
               STA   StepRep1        ; repeat count 1
               STA   StepRep2        ; repeat count 2
               RTS

               ; animate the sprites

DoSprites      LDA   StepDelay       ; check for timeout
               CMP   #0
               BEQ   DoSp0
               DEC   StepDelay
```

```
                RTS
DoSp0           LDX   StepIndex
                LDA   AnimData,X     ; check type
                CMP   #0             ; of animation
                BNE   DoSp1
                JMP   DoSp200
DoSp1           CMP   #1             ; type 1
                BNE   DoSp2
                LDA   AnimData+1,X   ; get frame
                STA   StepFrame
                LDA   AnimData+2,X   ; get current X
                STA   CurrX
                LDA   AnimData+3,X   ; get current Y
                STA   CurrY
                TXA
                CLC
                ADC   #6
                STA   StepIndex      ; point to next data
                LDX   StepFrame
                JSR   DrawSprite     ; draw current sprite
                JMP   DoSp0
DoSp2           CMP   #2             ; type 2
                BNE   DoSp3
                LDA   AnimData+2,X
                STA   CurrX
                LDA   AnimData+3,X
                STA   CurrY
                TXA
                CLC
                ADC   #6
                STA   StepIndex
                JMP   DoSp0
DoSp3           CMP   #3             ; type 3
                BNE   DoSp4
                LDA   Animdata+1,X
                STA   StepFrame
                LDA   AnimData+2,X
                CLC
                ADC   CurrX
                STA   CurrX
                LDA   AnimData+3,X
                CLC
                ADC   CurrY
                STA   CurrY
                LDA   AnimData+5,X
                STA   StepDelay
                TXA
                CLC
                ADC    #6
                STA    StepIndex
                LDA    #0
                STA    SprChrIdx
                LDX    StepFrame
                JSR    DrawSprite
                RTS
DoSp4           CMP    #4              ; type 4
                BNE    DoSp5
                LDA    StepRep1
                CMP    #0
                BNE    DoSp41
                LDA    AnimData+5,X
                STA    StepRep1
                JMP    DoSp42
DoSp41          DEC    StepRep1
DoSp42          LDA    StepRep1
                CMP    #0
                BEQ    DoSp45
                LDA    AnimData+4,X
                TAY
                LDX    #0
DoSp43          CPY    #0
                BEQ    DoSp44
                INX
                INX
                INX
                INX
                INX
                INX
                DEY
                JMP    DoSp43
DoSp44          STX    StepIndex
                JMP    DoSp0
DoSp45          TXA
                CLC
                ADC    #6
                STA    StepIndex
                JMP    DoSp0
DoSp5           CMP    #5
                BNE    DoSp200
                LDA    StepRep2
                CMP    #0
                BNE    DoSp51
                LDA    AnimData+5,X
                STA    StepRep2
                JMP    DoSp52
DoSp51          DEC    StepRep2
```

```
DoSp52          LDA   StepRep2
                CMP   #0
                BEQ   DoSp55
                LDA   AnimData+4,X
                TAY
                LDX   #0
DoSp53          CPY   #0
                BEQ   DoSp54
                INX
                INX
                INX
                INX
                INX
                INX
                DEY
                JMP   DoSp53
DoSp54          STX   StepIndex
                JMP   DoSp0
DoSp55          TXA
                CLC
                ADC   #6
                STA   StepIndex
                JMP   DoSp0
DoSp200         LDA   #4
                STA   CodeState
DoSp202         RTS

DrawSprite      LDA   #<FrameData
                STA   SprPtr            ; LSB of frame data
                LDA   #>FrameData
                STA   SprPtr+1          ; MSB of frame data
                DEX
Dsp0            CPX   #0
                BEQ   Dsp2
                LDA   #$6C        ; size of sprite definition
                CLC               ; block
                ADC   SprPtr
                STA   SprPtr
                BCC   Dsp1
                INC   SprPtr+1
Dsp1            DEX
                JMP   Dsp0
Dsp2            LDY   #$48
                STY   SprColIdx
                LDY   #$00
                STY   SprDatIdx
                STY   SprRow
                STY   SprColumn
                STY     SprRowOfs
                STY     SprColOfs
                LDX     SprChrIdx
Dsp10           LDY     SprDatIdx
                INY
                LDA     (SprPtr),Y
                DEY
                CMP     #$00
                BEQ     Dsp12
                INY
                INY
                STY     SprDatIdx
                JMP     Dsp15
Dsp12           LDA     CurrY
                CLC
                ADC     SprRowOfs
                STA     SpYLoc,X
                LDA     (SprPtr),Y
                STA     SpChar,X
                INY
                INY
                STY     SprDatIdx
                LDY     SprColIdx
                LDA     (SprPtr),Y
                STA     SpCtl,X
                LDA     CurrX
                CLC
                ADC     SprColOfs
                STA     SpXloc,X
Dsp14           INX
                INX
                INX
                INX
                STX     SprChrIdx
Dsp15           INC     SprColIdx
                LDA     SprColOfs
                CLC
                ADC     #8
                STA     SprColOfs
                INC     SprColumn
                LDA     SprColumn
                CMP     #6
                BEQ     Dsp16
                JMP     Dsp10
Dsp16           LDA     #0
                STA     SprColumn
                STA     SprColOfs
                LDA     SprRowOfs
                CLC
                ADC   #8
                STA   SprRowOfs
                INC   SprRow
                LDA   SprRow
                CMP   #6
                BCS   Dsp17
                JMP   Dsp10
Dsp17           RTS


                ; bank switching routine

SetBank         LDA   Banks,Y
                STA   Banks,Y
                RTS
Banks           DB    0,1,2,3,4,5,6,7

                ; This routine will transfer blocks
                ; from the database to their
                ; respective destinations.
                ; Blocks include sprite colors
                ; and sprite character definitions

LoadSprites     LDA   #<SpriteData   ; get starting address
                STA   Ind0           ; of data
                LDA   #>SpriteData
                STA   Ind0+1
Lscr0           LDA   PSTR           ; clear pending interrupts
                LDY   #$00
                LDA   (Ind0),Y       ; dest address low
                STA   Temp
                INY
                LDA   (Ind0),Y       ; dest address high
                STA   Temp+1
                AND   Temp           ; done if address
                CMP   #$FF           ; = $FFFF
                BEQ   LscrX
                INY
                LDA   Temp+1         ; get MSB of dest
                STA   PMAR
                LDA   Temp           ; get LSB of dest
                STA   PMAR
                LDA   #$00           ; get negative of
                SEC                  ; 16 bit data count
                SBC   (Ind0),Y
                STA   Temp
                INY
                LDA   #$00
                SBC   (Ind0),Y
                STA   Temp+1
                INY
Lscr1           LDA   (Ind0),Y      ; transfer actual data
                STA   PMDR
                INY
                BNE   Lscr2
                INC   Ind0+1        ; bump source
Lscr2           INC   Temp          ; check counter
                BNE   LScr1         ; for more data
                INC   Temp+1
                BNE   LScr1
                CLC                 ; update pointer
                TYA                 ; to next data block
                ADC   Ind0
                STA   Ind0
                BCC   Lscr4
                INC   Ind0+1
Lscr4           JMP   Lscr0
LscrX           RTS

                ; clear the screen
                ; fill with character $FF

ClrScreen       LDA   #$20          ; dest MSB = $20
                STA   PMAR
                LDA   #0            ; dest LSB = $00
                STA   PMAR
                LDX   #$00
                LDY   #$00
                LDA   #$FF
ClrScreen0      STA   PMDR          ; store character
                DEX                 ; $400 times
                BNE   ClrScreen0
                INY
                CPY   #$04
                BNE   ClrScreen0
                RTS

                ; clear the background
                ; character set area

ClrChars        LDA   #$10
                STA   PMAR
                LDA   #$00
                STA   PMAR
                LDA   #0
                TAX
                TAY
ClrCh           STA   PMDR
                INX
                BNE   ClrCh
                LDA   PSTR
                INY
                CPY   #$10
                BNE   ClrCh
                RTS

                ; clear the sprite
                ; character area

ClrSprites      LDA   #$00
                STA   PMAR
                LDA   #$00
                STA   PMAR
                LDA   #0
                TAX
                TAY
ClrSp           STA   PMDR
                INX   .
                BNE   ClrSp
                INY
                CPY   #$10
                BNE   ClrSp
                RTS

SpriteData      ; the first block contains the sprite
                ; color sets
                DW    $3F10          ; destination address
                DW    16             ; data count
                DB    1,2,3,4        ; total of 16 color
                                     ; definition bytes

                ; the second block contains the sprite
                ; character set definition
                DW    $0000          ; destination address
                DW    $1000          ; data count
                DB    1,2,3,4        ; total of 4096 character
                                     ; definition bytes.

                ; terminator
                DW    $FFFF

                ; 6502 hardware vectors
```

```
                    ORG     $FFFA

                    DW      NMIVector       ; NMI service routine
                    DW      RESVector       ; Reset entry routine
                    DW      IRQVector       ; IRQ service routine

                    END
```

## SCREEN BLANKING

The PPU Control Register 2 (PCR2, $2001) controls several functions. Bit 0, when set, turns the display to monochrome Bit3, when set blanks the background. Bit 4, when set, blanks the sprites. For the CPU to alter background or sprites, the respective blanking bit must be set. It seems that writes to PCR1 have no effect when either of the blanking bits are set.

The purpose of the blanking bits is thus twofold. They may be used to blank a screen when building a screen to prevent screen jumping and distortion, since the screen becomes unstable when accessed during the visible portion of the frame. They must also be set to allow CPU access to the video memory and registers.

```
PCR2            EQU  $2001
ShPCR2          EQU  $FE

NMIEntry        LDA  ShPCR2     ; get shadow
                ORA  #$18       ; blank bg/sprites
                STA  ShPCR2     ; save shadow
                STA  PCR1       ; store register

                ; do all video reads/writes until
                ; visible portion of screen is reached.
                ; You can transfer about 100 characters
                ; to the screen during this time

NMICont         LDA  ShPCR2     ; get shadow
                AND  #$E7       ; enable bg/sprites
                STA  ShPCR2     ; save shadow
                STA  PCR1       ; store register

                ; the screen becomes active and will
                ; not be disturbed. Game logic should
                ; be handled here.
```

## NMI CONTROL

The CPU has not internal NMI generation. NMIs are generated by the PPU at a 60Hz rate. Most NES games use the NMI signal to synchronize the entire program execution to. NMI recognition cannot be disabled at the CPU, but NMI generation can disabled at the PPU.

Register PCR1, bit 7 is the interrupt enable bit. NMI's are generated when bit 7 = 1. This bit should be reset upon entry into VBlank and set again upon exit. This code is commonly found in NES games:

```
PCR1            EQU  $2000
ShPCR1          EQU  $FF

NMIEntry        LDA  ShPCR1      ; get shadow
                AND  #$7F        ; reset intr enable
                STA  ShPCR1      ; save shadow
                STA  PCR1        ; store register

NMIExit         LDA  ShPCR1      ; get shadow
                ORA  #$80        ; set intr enable
                STA  ShPCR1      ; save shadow
                STA  PCR1        ; store register
```

The PPU Status Register (PSTR, $2002) is a read only register. This register contains the NMI occurred flag. It must be read after every interrupt to re-enable interrupts. Reading this register not only clears the interrupt occurred bit but also seems to reset the scroll registers. Since you have to read this register everytime you execute VBlank you must also reset your scrolling registers every time you go through VBlank. The register is accessed by simply reading it.

All NES games use the interrupt bit in the initialization code to let the hardware stabilize. The code waits for 2 or 3 interrupts to occur before any action is taken. The interrupt bit is simply polled. The wait loops look as follows:

```
PSTR       EQU  $2002

Wait1      LDA  PSTR
           BPL  Wait1       ; Loop while bit 7 reset
Wait2      LDA  PSTR
           BPL  Wait2
Wait3      LDA  PSTR
           BPL  Wait3
```

## IRQ CONTROL

The CPU does generate internal IRQs. These are logically wire-ored to the external IRQ pin. The PPU does not generate IRQs. MMC3 cartridges may be programmed to generate an IRQ. In all other environments the external IRQ pin on the CPU is left unconnected.

An internal IRQ is generated at the end of any CPU DMA cycle. DMA cycles include sprite parameter passing form CPU to PPU and sampled sound output. IRQ generation can be disabled. IRQ recognition is controlled through the 6502 instructions CLI and SEI.

The source of an internal IRQ can be identified as follows:

```
VMER        $4015       (Read)
                        Bit 7 = 1 sample sound done
                        Bit 6 = 1 sprite transfer done
```

IRQ generation can be enabled as follows:

```
JOY2PORT    $4017       (Write)
                        Bit 7 = 1 (always)
                        Bit 6 = 0 IRQ enable
```

## GAME CONTROLLERS

The NES has two game controller ports. The most commonly used game controllers are the joystick controllers supplied with each console. The controllers contain up to eight switches.

When controllers are to be read first a load command is issued by the CPU (JOY1PORT, $4016 and JOY2PORT, $4017). The load command then transfers the individual switch states into an 8 bit shift register in the controller. This is followed by 8 successive reads of the controller port to get all 8 status bits into the processor. Since the controllers are simple digital switches the resulting data represents simple on/off information for each switch.

Following is a code example of a joystick read routine. The routine, which should be called once every frame, reads the 8 data bits and assembles a command word. It further generates a debounced output and has a facility for autorepeat.

```
                ; this routine will read controller A

JOYBUTA         EQU     %10000000       ; 'A' Button
JOYBUTB         EQU     %01000000       ; 'B' Button
JOYSEL          EQU     %00100000       ; Select
JOYSTART        EQU     %00010000       ; Start
JOYUP           EQU     %00001000       ; Up
JOYDOWN         EQU     %00000100       ; Down
JOYLEFT         EQU     %00000010       ; Left
JOYRIGHT        EQU     %00000001       ; Right


Joy1Port        EQU     $4016           ; Controller A port
```

```
Joy2Port      EQU   $4017        ; Controller B port

Joy1Last      EQU   $00          ; Ctrl A last value
Joy1Data      EQU   $01          ; Ctrl A current data
Joy1Edge      EQU   $02          ; Ctrl A debounced data
Joy1Delay     EQU   $03          ; Ctrl A repeat delay

              ; Controller read routine with auto repeat
              ; Should be called from within vblank

ReadJoy       LDY   #1           ; load controller 1 shift
              STY   JOY1PORT     ; shift register
              DEY
              STY   JOY1PORT
              LDY   #8           ; number of bits
ReadJ1        PHA                ; save current new bits
              LDA   JOY1PORT     ; get next bit(s)
              STA   Joy1Last
              LSR   A            ; shift down second bit
              ORA   Joy1Last     ; fold in first bit
              LSR   A            ; new bit into carry
              PLA                ; restore new bits
              ROL   A            ; shift in new bit
              DEY                ; dec bit counter
              BNE   ReadJ1
              LDY   Joy1Data     ; transfer previous data
              STY   Joy1Last
              STA   Joy1Data     ; save new data
              EOR   Joy1Last     ; debounce it
              AND   Joy1Data
              STA   Joy1Edge
              LDY   #$30         ; delay until repeat
              LDA   Joy1Data     ; see if new data
              CMP   Joy1Last
              BNE   ReadJ2       ; start new delay
              DEC   Joy1Delay
              BNE   ReadJ3
              STA   Joy1Edge     ; store repeat value
              LDY   #10          ; delay between repeats
ReadJ2        STY   Joy1Delay
ReadJ3        RTS
```

I have also seen other types of controllers. The game 'Arkanoid' requires a paddle controller and is packaged with a dedicated controller (It can be played with a paddle, but this is somewhat awkward). This controller performs A/D conversion of the current paddle potentiometer position and generates digital position data.

The light gun consists of a light sensitive photocell which kicks in when a certain threshold of light intensity is exceeded.

Other controllers with auto fire repeat contain timing circuits that periodically send new button presses whenever a certain button is kept depressed.

The controller ports can be viewed as general purpose eight bit input only ports.

## SOUND PROGRAMMING

Sound generating hardware is located in the CPU. There are several different voices, each individually programmable. Each voice functions differently.

## VOICE 1

Voice 1 generates a square wave output. Amplitude, frequency and duty cycle are programmable.

## VOICE 1 FREQUENCY

The frequency is set by writing an 11 bit value to V1R3 ($4002, BITS 0 - 7, LSB) and V1R4 ($4003, bits 0 - 2, MSB). A seven octave frequency table follows:

| Octave | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| C | $6AD | $356 | $1AA | $0D4 | $06A | $035 | $01A |
| C# | $64A | $325 | $192 | $0C9 | $064 | $031 | $019 |
| D | $5EF | $2F8 | $17B | $0BD | $05E | $02E | $018 |
| D# | $59B | $2CD | $166 | $0B3 | $059 | $02C | $016 |
| E | $54B | $2A5 | $152 | $0A9 | $054 | $029 | $015 |
| F | $4FE | $27F | $13F | $09F | $04F | $027 | $014 |
| F# | $4B5 | $25B | $12D | $096 | $04A | $025 | $013 |
| G | $473 | $23A | $11C | $08E | $047 | $023 | $012 |
| G# | $430 | $219 | $10C | $086 | $042 | $021 | $011 |
| A | $3F5 | $1FB | $0FD | $07E | $03E | $01F | $010 |
| A# | $3BD | $1DE | $0EF | $077 | $03B | $01D | $00F |
| B | $387 | $1C3 | $0E1 | $070 | $038 | $01B | $00E |

## VOICE 1 DUTY CYCLE

The duty cycle of the square wave output is selectable from four preset values. The duty cycle will alter the timbre of the sound. The closer you get to a 50% duty cycle, the fuller the sound will be. A 50% duty cycle will sound like a sine wave, whereas a lower duty cycle will produce the raspy sound of a sawtooth. The duty cycle is expressed as a percentage of the square wave being at a '1' level versus the entire period.

The duty cycle is selected by setting V1R1 ($4000, bits 7,6). Values are as follows:

| B7 | B6 | Duty Cycle(%) |
|---|---|---|
| 0 | 0 | 12.5 |
| 0 | 1 | 25.0 |
| 1 | 0 | 50.0 |
| 1 | 1 | 75.0 |

## VOICE 1 SOUND LENGTH

Sound lengths may be programmed in hardware or controlled through software. The length mode is specified by V1R1 bit 5. If bit 5 = 1 the sound is continuous, otherwise a length counter is employed.

The length counter occupies the upper 5 bits of V1R4. It holds a 5 bit code that specifies an 8 bit count in jiffies (A jiffy is 1/60th of a second). When V1R1 bit 5 is reset the following length codes become active:

| Code (Hex) | Length (Jiffies, Decimal) |
|---|---|
| 00 | 5 |
| 01 | 126 |
| 02 | 10 |
| 03 | 1 |
| 04 | 19 |
| 05 | 2 |
| 06 | 40 |
| 07 | 3 |
| 08 | 80 |
| 09 | 04 |
| 0A | 30 |
| 0B | 5 |
| 0C | 7 |
| 0D | 6 |
| 0E | 13 |
| 0F | 7 |
| 10 | 6 |
| 11 | 8 |
| 12 | 12 |
| 13 | 9 |
| 14 | 24 |
| 15 | 10 |
| 16 | 48 |
| 17 | 11 |
| 18 | 96 |
| 19 | 12 |
| 1A | 36 |
| 1B | 13 |

| | |
|---|---|
| 1C | 8 |
| 1D | 14 |
| 1E | 16 |
| 1F | 15 |

This table does not strike me as being very useful. I have very rarely seen a program make use of hardware lengths. You may derive a set of standard note durations from the above table, but you can't implement tempo changes. It therefore seems advisable to create your own duration tables and let the software keep track of note lengths.

## VOICE 1 AMPLITUDE/ENVELOPE

Amplitude can be specified by writing to V1R1, bits 0 - 3. For this to work properly bit 4 has to be set. The amplitude can be directly written consisting of a 4 bit value, where $0F is maximum amplitude. When bit 4 is reset the wave generator goes into some kind of hardware envelope control. The exact mode of operation has not yet been determined.

## VOICE 1 FREQUENCY SWEEP

V1R2 allows a hardware frequency sweep. Bit 7, when set, activates frequency sweep. Bit 4 is the sweep direction, 0 decreases the pitch and 1 increases the pitch. Bits 2 - 0 control the pitch speed. 0 is the slowest delay (fastest sweep) and 7 is the longest delay (slowest sweep).

In summary the voice 1 registers function as follows:

```
V1R1        $4000       DDLMAAAA, where
                        D = duty cycle selection
                        L = Sound length mode
                        M = amplitude/envelope mode
                        A = Amplitude value

V1R2        $4001       OxxxDCCC, where
                        O = frequency sweep enable
                        D = sweep direction
                        C = sweep delay

V1R3        $4002       FFFFFFFF, where
                        F = frequency lower 8 bits

V1R4        $4003       LLLLLFFF, where
                        F = frequency upper 3 bits
                        L = length code
```

Voice 1 can be enabled by setting bit 0 of the Voice Master Enable Register (VMER, $4015) and disabled by resetting bit 0.

## VOICE 2

Voice 2 functions identically to Voice 1. The entire discussion for Voice 1 is valid for voice 2 except for reassignment of the registers.

The Voice 2 registers function as follows:

```
V2R1        $4004       DDLMAAAA, where
                        D = duty cycle selection
                        L = Sound length mode
                        M = amplitude/envelope mode
                        A = Amplitude value

V2R2        $4005       OxxxDCCC, where
                        O = frequency sweep enable
                        D = sweep direction
                        C = sweep delay

V2R3        $4006       FFFFFFFF, where
                        F = frequency lower 8 bits

V2R4        $4007       LLLLLFFF, where
                        F = frequency upper 3 bits
                        L = length code
```

Voice 2 can be enabled by setting bit 1 of the Voice Master Enable Register (VMER, $4015) and disabled by resetting bit 1.

## VOICE 3

Voice 3 is a triangle wave sound generator. It includes the features of Voices 1/2, except frequency sweep. Its output is muted and has accompanying noise.

## VOICE 3 LENGTH

Voice 3 length is specified the same way that sound length is specified for voices 1/2, except the hardware length enable bit is bit 7, V3R1 ($4008). It also seems as if the lower 7 bits of V3R1 can be used to directly specify a 7 bit jiffy count, as opposed to using codes. This length counter becomes active when the alternate length counter (V3R4) is assigned a decoded value less than the value in V3R1. The exact purpose of this setup is unclear for me.

The Voice 3 registers function as follows:

| | | |
|---|---|---|
| V3R1 | $4008 | ELLLLLLL<br>E = Sound length mode<br>L = sound length setting |
| V3R2 | $4009 | Not Used |
| V3R3 | $400A | FFFFFFFF, where<br>F = frequency lower 8 bits |
| V3R4 | $400B | LLLLLFFF, where<br>F = frequency upper 3 bits<br>L = length code |

Voice 3 can be enabled by setting bit 2 of the Voice Master Enable Register (VMER, $4015) and disabled by resetting bit 2.

## VOICE 4

Voice 4 is a pseudo random noise generator. It duplicates the sound length and amplitude functions of Voices 1/2.

## VOICE 4 FREQUENCY

Even though the frequency of a noise generator is random, there seems to be a way to change the frequency range of the noise output. V4R3, lower nibble, seems to change the sound characteristics of the resulting output, the exact function or frequency range produced by the discreet value used is unknown to me.

The Voice 4 registers function as follows:

| | | |
|---|---|---|
| V4R1 | $400C | xxLMAAAA, where<br>L = Sound length mode<br>M = amplitude/envelope mode<br>A = Amplitude value |
| V4R2 | $400D | Not Used |
| V4R3 | $400E | xxxxFFFF, where<br>F = noise frequency range |
| V4R4 | $400F | LLLLLxxx, where<br>L = length code |

Voice 4 can be enabled by setting bit 3 of the Voice Master Enable Register (VMER, $4015) and disabled by resetting bit 3.

## VOICE 5

Voice 5 is a D/A channel used to playback sampled sounds. There are two methods available to play back digitized sound.

The first method uses a straightforward approach in which you write a byte at a time to a register which immediately converts the written digital value to an analog value and sends it out through the sound outputs. This method requires you to periodically write data at the desired sample rate.

The second method is fully automatic, but it uses delta modulation. In delta modulation each new value does not replace the old one but rather is used to modify the old value. In this particular implementation data is clocked in at a predetermined sample clock rate. At each tick of the clock the next available bit determines how the existing D/A value is modified. If the bit is set the existing value is incriminated, otherwise the existing value is decremented.

It follows that your raw digital data must be in a form in which each byte differs from the preceding one by one, either plus or minus. You then build an array of bits representing the difference (1 = +1, 0 = -1) between each successive byte. Now put eight bits into a byte, with the most significant bit in bit position 7. You now have an array of packed deltas, eight deltas per byte. This represents the digital data as required by the delta mod circuit.

To prepare for sound output you load registers with the sample length (number of packed bytes), the sample start (a memory location) and the initial value of the D/A converter. To start sound you set bit 4 of VMAR ($4015) to 1.

The sample will play and sound output will terminate when the end of the sample is reached. Automatic repeat is also available, in which case you have to stop sound output by resetting bit 4 in VMAR. Upon completion of a sample an internal IRQ will be generated, if enabled.

A sample output is generated by checking the current bit at every sample clock tick and incrementing or decrementing the current D/A value. A new byte of delta bits is fetched every 8 sample clock ticks.

## VOICE 5 REGISTERS

Voice 5 registers function as follows:

```
V5R1        $4010       IRxxSSSS where
                        I = IRQ enable
                        R = repeat enable
```

| | | |
|---|---|---|
| | | S = sample clock frequency |
| V5R2 | $4011 | DDDDDDDD where<br>D = Direct data register. This data is directly output to the D/A converter. |
| V5R3 | $4012 | AAAAAAAA where<br>A = Delta mod address pointer. Should be initialized to point at beginning of data. Note that this represents bits 6 through 13 of the true address. Bits 0 through 5 are assumed to be 0 and bits 13 and 14 are assumed to be 1. |
| V5R4 | $4013 | CCCCCCCC where<br>C = 8 bit count of delta mod bytes. Note that this represents bits 4 through 11 of the true count. The lower nibble is always set to 0. |

| | |
|---|---|
| 0000 | 0 |
| 0001 | 1 |
| 0010 | 2 |
| 011 | 3 |
| 0100 | 4 |
| 0101 | 5 |
| 0110 | 6 |
| 0111 | 7 |
| 1000 | 8 |
| 1001 | 9 |
| 1010 | A |
| 1011 | B |
| 1100 | C |
| 1101 | D |
| 1110 | E |
| 1111 | F |

0001 = 1

0010 = 2

0100 = 4

1000 = 8

1110 = E

1101 = D

1011 = B

0111 = 7

## CARTRIDGE TYPES

I have come across several different types of cartridges, which I will list and describe. This list is not exhaustive, since I haven't looked at all available cartridges.

On the program side the NES decodes memory space $8000 - $FFFF in which almost all cartridge activities take place. Exceptions are cartridges that allow external program memory ($6000 - $7FFF). The NES, however, does not generate any control signals to decode this space. The logic for doing so must be on the cartridge.

On the character side the NES decodes and assigns memory space $0000 - $1FFF, which is used for character and sprite character set definition. On some cartridges part of the memory may be used as extra screen memory. Again the logic for performing these tasks must be on the cartridge.

The cartridge also decides whether to set scrolling mode to horizontal or vertical. On the more basic types this is achieved through solder shorts. On other types cartridge logic allows software commands to select the scrolling mode.

Some cartridges have optional battery backup for the external program memory only. This is used in games like Zelda to save the game environment for later continuation. The scheme is slightly unreliable. If the NES gets powered up without pressing the reset button simultaneously (as the warning states on such carts) the processor might spuriously write to external memory, destroying saved game information.

## TYPE 1 CARTRIDGES

These are the most basic type cartridges. The may have up to 32K of program ROM permanently mapped to CPU $8000 - $FFFF. They have 8K of character ROM/RAM mapped to PPU $0000 - $1FFF. The scrolling mode is selected by a solder short on the cart.

Type 1 carts include Duckhunt, Elevator Action, Golf, Gyromite, Kung Fu, Mario Bros., Pinball, Super Mario Bros., Excite Bike, Urban Champion, Galaga, Xevious and Pac-Man.

A variation of Type 1 carts allows you up to 32K (4 banks) of character ROM.

Carts using this variation include Arkanoid, Spy Hunter, Gotcha, Mickey Mousecapades, Legend Of Kage, Track And Field, Donkey Kong/Junior, Gradius and Starforce.

The character bank is selected by writing a value from 0 to 3 to any address in program memory. To avoid bus contention, the value to be written should first be placed on the data bus by a read statement followed immediately by a write.

A general purpose bank select routine to achieve this could be written as:

```
          . .
          . .
          LDA   #1
          JSR   SetCBank
          . .
          . .

SetCBank  TAY                ; transfer bank to Y
          LDA   CBanks,Y     ; put value on data bus
          STA   $8000        ; select bank
          RTS
CBanks    DB    0,1,2,3
```

## TYPE 2 CARTRIDGES

This type offer program memory bank selection in 16K units and a single 8K character ROM. The scrolling mode is selected by a solder short on the cart.

Type 2 carts include Commando, Double Dribble, Castlevania, Ghost's-n-Goblins, Rush'n Attack, Ikari Warriors, The Goonies II, Jackal, Wrestling, Blades Of Steel, Megaman, Trojan, Legendary Wings, Stinger, Gunsmoke, Contra, Super Pitfall and 3D World Runner.

Program banks are mapped as follows. Special logic on the cart permanently maps bank 7 of program memory to CPU area $C000 - $FFFF. The 6502 vectors and initialization code only have to reside in bank 7. The remaining program area, $8000 - $BFFF can contain any of the 8 banks, even though replicating bank 7 into the lower area doesn't make much sense.

Bank selection is subject to the same restrictions as apply for Type 1 carts. The previous bank selection routine has to be slightly altered as follows:

```
            . .
            . .
            LDA   #1
            JSR   SetPBank
            . .
            . .

SetPBank    TAY                ; transfer bank to Y
            LDA   PBanks,Y     ; put value on data bus
            STA   $8000        ; select bank
            RTS
PBanks      DB    0,1,2,3,4,5,6,7
```

## HYBRID TYPES

Even though I haven't come across any I would imagine there are carts that allow both program and character bank selection. Since the LS161 latch on the cartridge is 4 bits wide you could assign 2 bits to character bank selection and 2 bits to program bank selection, or any combination requiring a total of 4 bits.

Type 3 cartridge.

## MULTI MEMORY CONTROLLER 1

The Multi Memory Controller 1 (MMC1) is a PAL installed on cartridges that allows program and character bank selection and size setting. It also allows for H/V selection through software and enables a program to use cartridge character memory of 2 video screens for a total of 4 video screens, allowing smooth diagonal scrolling. Finally the MMC1 will support exterior program RAM.

Cartridges using the MMC1 include Zelda, Zelda II, Rad Racer, Metroid, Kid Icarus, Double Dragon, Cobra, Castlevania II, Karnov, Fighting Golf, Ninja Gaiden, Blaster Master, Double Dragon and Xenophobe.

The MMC1 allows a program address space of 256K Bytes (PA0 - PA17). Program banks may be 32K/per bank, yielding 8 banks maximum, or 16K/per bank, yielding 16 banks maximum. The MMC1 controls which bank of planar memory is active at any time. Because the MMC1 does not generate a fixed program control area reset vectors and code must be located in each 16K segment to assure proper startup.

External program RAM is supported (8K Bytes mapped at $6000 - $7FFF). Provisions are made for optional battery backup to retain game parameters in external program RAM.

Character memory may be either RAM or ROM. An address space of 128K Bytes is supported (CA0 - CA16). Character banks may be 4K/per bank, yielding 32 banks maximum, or 8K/per bank, yielding 16 banks maximum. When 4K banks are selected the two banks don't have to be continuous.

Screen memory selection is through software. See the MMC1 register description for functional details.

The MMC1 is programmed by writing to specific areas in program memory. Since the MMC1 has only two data inputs data must be sent bit by bit through software. Active bits are D7 (control) and D0 (data), therefore during a data transfer only these two bits are significant.

When D7 is set the MMC1 is reset, most likely initializing the internal serial to parallel converter.

To reset the MMC1 the following code could be used:

```
LDA   #$80       ; bit 7 set to initialize
                 ; bit 0 not used
STA   $8000      ; initialize MMC1, any address
                 ;  should do
```

The MMC1 expects 5 bits of data for every command, therefore five separate writes are required to transfer the data.

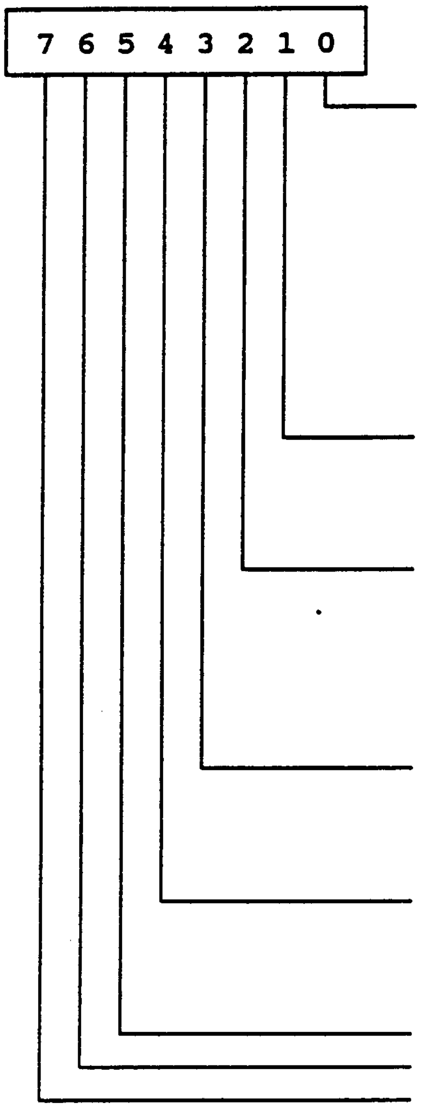
To transfer a datum to the MMC1 the following code could be used:

```
LDA   #$15       ; bits 0 - 4 data, bit 7 0
STA   $8000      ; first bit
LSR   A
STA   $8000      ; second bit
LSR   A
STA   $8000      ; third bit
LSR   A
STA   $8000      ; fourth bit
LSR   A
STA   $8000      ; fifth bit
```

Following is a description of each MMC1 register.

## MMC1 REGISTER 0

Register 0 is accessed by writing to location $8000.



R0D0: Screen Select.
When R0D1 is in Single Screen Mode
R0D0 selects the active screen.
0 = $2000 - $23FF.
1 = $2400 - $27FF.
When R0D1 is in 4 Screen Mode
R0D0 selects the scrolling mode.
0 = Horizontal.
1 = Vertical.

R0D1: Screen Mode.
0 = Single Screen Mode.
1 = 4 Screen Mode.

R0D2: Program Area Mapping.
When R0D3 is in 32K mode R0D3
is ignored. Otherwise
0 = $8000 - $BFFF.
1 = $C000 - $FFFF.

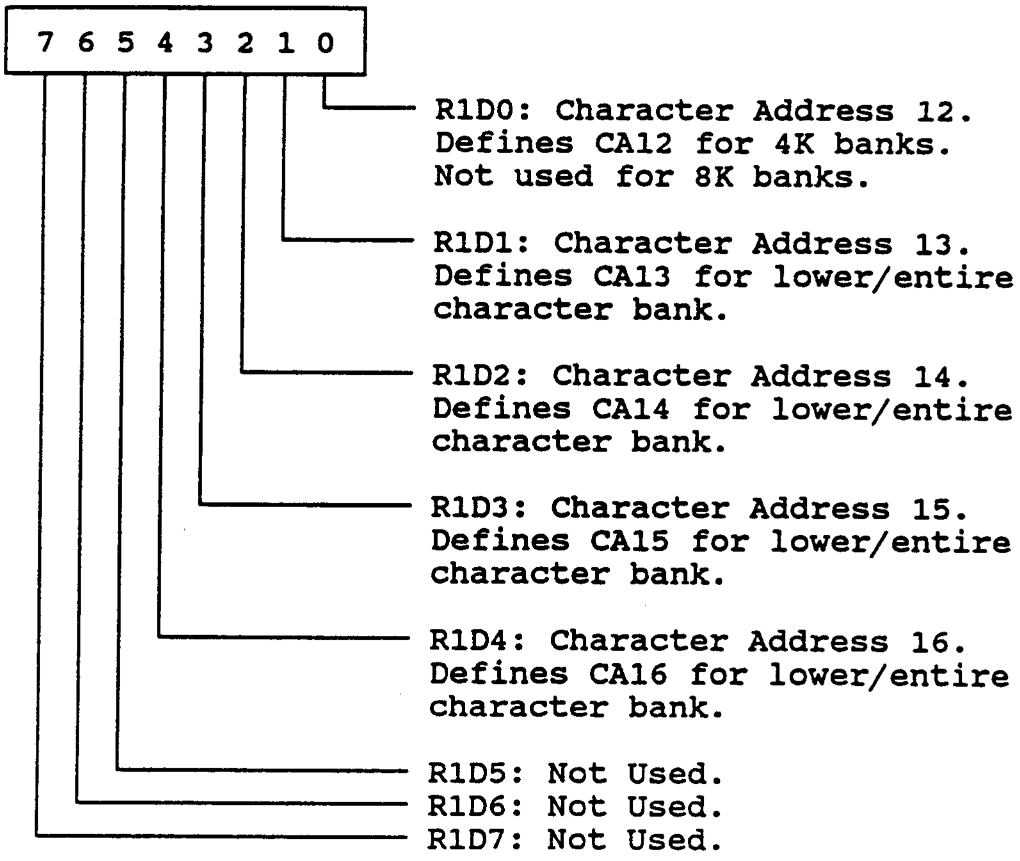
R0D3: Program Bank Size.
0 = 32K/per bank.
1 = 16K/per bank.

R0D4: Character Bank Size.
0 = 8K/per bank,
1 = 4K/per bank.

R0D5: Not Used.
R0D6: Not Used.
R0D7: Not Used.

## MMC1 REGISTER 1

Register 1 is accessed by writing to location $A000.
If R0D4 is set to 4K/per bank register 1 selects the lower character bank (PPU $0000 - $0FFF).
IF R0D4 is set to 8K/per bank register 1 selects the entire character bank (PPU $0000 - $1FFF).



R1D0: Character Address 12.
Defines CA12 for 4K banks.
Not used for 8K banks.

R1D1: Character Address 13.
Defines CA13 for lower/entire character bank.

R1D2: Character Address 14.
Defines CA14 for lower/entire character bank.

R1D3: Character Address 15.
Defines CA15 for lower/entire character bank.

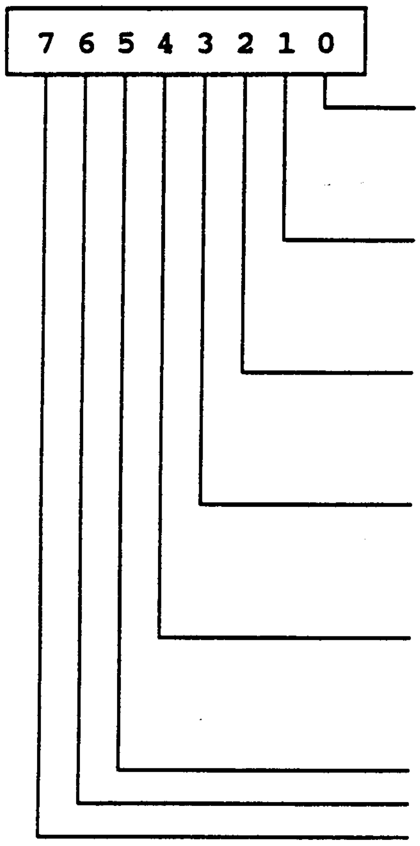
R1D4: Character Address 16.
Defines CA16 for lower/entire character bank.

R1D5: Not Used.
R1D6: Not Used.
R1D7: Not Used.

2

## MMC1 REGISTER 2

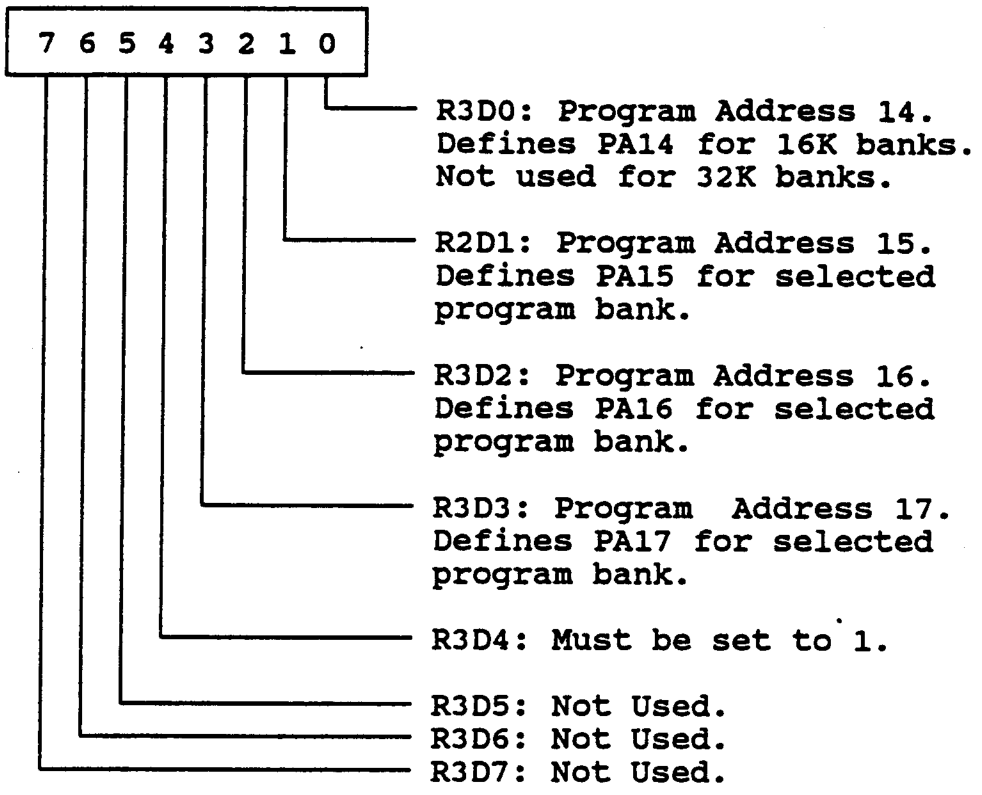
Register 1 is accessed by writing to location $C000.
If R0D4 is set to 4K/per bank register 2 selects the upper character bank (PPU $1000 - $1FFF).
IF R0D4 is set to 8K/per bank register 2 is not used.



```
R2D0: Character Address 12.
Defines CA12 for upper
character bank.

R2D1: Character Address 13.
Defines CA13 for upper
character bank.

R2D2: Character Address 14.
Defines CA14 for upper
character bank.

R2D3: Character Address 15.
Defines CA15 for upper
character bank.

R2D4: Character Address 16.
Defines CA16 for upper
character bank.

R2D5: Not Used.
R2D6: Not Used.
R2D7: Not Used.
```

## MMC1 REGISTER 3

Register 3 is accessed by writing to location $E000.



R3D0: Program Address 14.
Defines PA14 for 16K banks.
Not used for 32K banks.

R2D1: Program Address 15.
Defines PA15 for selected
program bank.

R3D2: Program Address 16.
Defines PA16 for selected
program bank.

R3D3: Program Address 17.
Defines PA17 for selected
program bank.

R3D4: Must be set to 1.

R3D5: Not Used.
R3D6: Not Used.
R3D7: Not Used.

## MULTI MEMORY CONTROLLER 2

The Multi Memory Controller 2 (MMC2) is a PAL installed on cartridges allowing character bank, program bank and scrolling mode selection through software.

Memory sizes supported are 128K bytes of program ROM and 128K Bytes of character ROM.

Cartridges using the MMC2 include Mike Tyson's Punch Out.

## MULTI MEMORY CONTROLLER 3

The Multi Memory Controller 3 (MMC3) is a custom chip installed on cartridges that allows program and character bank selection and size setting. It also allows for H/V selection through software. The MMC3 has an internal timer that can be set to generated CPU IRQ's. Finally the MMC3 will support extra program RAM.

Cartridges using the MMC3 include Super Mario Bros. 2.

The MMC3 allows a maximum program address space of 512K Bytes (PA0 - PA18). The CPU cartridge 32K memory area is mapped into 4 8K areas.

External program RAM is supported (8K Bytes mapped at $6000 - $7FFF).

Character memory may be either RAM or ROM. An address space of 256K Bytes is supported (CA0 - CA17). Character banks are sized to 2K per bank and 1K per bank. $E000 ~ [illegible] disable Internal Timer

The following write only registers are used to pass commands to the MMC3.

| | | |
|---|---|---|
| $A000 | MSSR | Scroll selection register. Bit 0 selects horizontal (0) and vertical (1) scrolling. |
| $A001 | MWPR | Write Protect register. Bit 7 selects write protect (1) and write enable (0). |
| $8000 | MBRS | Bank register select register. This register selects one of 7 registers that are used to bank select program and character banks. |
| $8001 | MBRV | Bank select value register. Writing to this register will write a value to the register selected by MBRS. |

| | |
|---|---|
| MBRS = $80 | Character Bank Select Register 0 (CBSR0) |
| MBRS = $81 | Character Bank Select Register 1 (CBSR1) |
| MBRS = $82 | Character Bank Select Register 2 (CBRS2) |
| MBRS = $83 | Character Bank Select Register 3 (CBSR3) |
| MBRS = $84 | Character Bank Select Register 4 (CBSR4) |
| MBRS = $85 | Character Bank Select Register 5 (CBRS5) |
| MBRS = $86 | Program Bank Select Register 0 (PBSR0) |
| MBRS = $87 | Program Bank Select Register 1 (PBSR1) |

## MMC3 PROGRAM BANK SELECTION

The MMC3 supports 512K of program memory. The cartridge program area ($8000 - $FFFF) is divided into four areas of 8K each. For 512K this gives a total of 64 banks.

| | |
|---|---|
| Area 1 | $8000 - $9FFF |
| Area 2 | $A000 - $BFFF |
| Area 3 | $C000 - $DFFF |
| Area 4 | $E000 - $FFFF |

The memory for area 1 can be bank switched to any 8K segment. Bank selection is done through PBSR0. Since 64 banks require 6 bits, only bits 5 through 0 of PBSR0 are used, Bits 7 and 6 are ignored.

The memory for area 2 can be bank switched to any 8K segment. Bank selection is done through PBSR1. Since 64 banks require 6 bits, only bits 5 through 0 of PBSR1 are used, Bits 7 and 6 are ignored.

The memory for area 3 is permanently mapped to bank 63, memory address $7C000 - $7DFFF. Note that for games with smaller program ROMs (Super Mario II, 128K) the upper two program address bits are not used and area 3 automatically becomes mapped to bank 15, memory address $1C000 - $1DFFF.

The memory for area 4 is permanently mapped to bank 64, memory address $7E000 - $7FFFF. This is the area that should include all 6502 hard vectors. Note that for games with smaller program ROMs (Super Mario II, 128K) the upper two program address bits are not used and area 3 automatically becomes mapped to bank 16, memory address $1D000 - $1FFFF.

## MMC3 PROGRAM BANK SELECTION EXAMPLE

Assume it is desired to set up program mapping as follows:

```
Area 1 to bank 61, memory address $78000 - $79FFF
Area 2 to bank 62, memory address $7A000 - $7BFFF

PrgBankSel      LDY  #7
Bs1             TYA
                ORA  #$80
                STA  $8000          ; do reg select
                LDA  BankVals,Y     ; get bank values
                STA  $8001          ; save value
                DEY
                CPY  #5
                BCS  Bs1
                RTS
```

[illegible]

```
BankVals        DB    0,0,0,0,0,0,$3C,$3D
```

## MMC3 CHARACTER BANK SELECTION

The MMC3 supports a maximum of 256K of character memory. The cartridge character area ($0000 - $1FFF) is divided into 6 areas. The first two areas are 2K in size, allowing one of 128 banks to be selected, requiring 6 bits for bank selection. The remaining areas are 1K in size, allowing one of 256 banks to be selected, requiring 7 bits for bank selection.

```
Area 1    $0000 - $07FF  Set 1 Character $00 - $7F
Area 2    $0800 - $0FFF  Set 1 Character $80 - $FF
Area 3    $1000 - $13FF  Set 2 Character $00 - $3F
Area 4    $1400 - $17FF  Set 2 Character $40 - $7F
Area 5    $1800 - $1BFF  Set 2 Character $80 - $BF
Area 6    $1C00 - $1FFF  Set 2 Character $C0 - $FF
```

The memory for area 1 can be bank switched to any 2K segment. Bank selection is done through CBSR0. Bits 7 through 1 are used to select one of 128 banks. Bit 0 is ignored.

The memory for area 2 can be bank switched to any 2K segment. Bank selection is done through CBSR1. Bits 7 through 1 are used to select one of 128 banks. Bit 0 is ignored.

The memory for area 3 can be bank switched to any 1K segment. Bank selection is done through CBSR2. Bits 7 through 0 are used to select one of 256 banks.

The memory for area 4 can be bank switched to any 1K segment. Bank selection is done through CBSR3. Bits 7 through 0 are used to select one of 256 banks.

The memory for area 5 can be bank switched to any 1K segment. Bank selection is done through CBSR4. Bits 7 through 0 are used to select one of 256 banks.

The memory for area 6 can be bank switched to any 1K segment. Bank selection is done through CBSR5. Bits 7 through 0 are used to select one of 256 banks.

## MMC3 CHARACTER BANK SELECTION EXAMPLE

Assume it is desired to set up character mapping as follows:

```
Area 1 to 2K bank 16, memory address $08000 - $087FF
Area 2 to 2K bank 17, memory address $08800 - $08FFF
Area 3 to 1K bank 36, memory address $09000 - $093FF
Area 4 to 1K bank 37, memory address $09400 - $097FF
Area 5 to 1K bank 38, memory address $09800 - $09BFF
Area 6 to 1K bank 39, memory address $09C00 - $09FFF

ChrBankSel      LDY  #5
Bs1             TYA
                ORA  #$80
                STA  $8000          ; do reg select
                LDA  BankVals,Y     ; get bank values
                STA  $8001          ; save value
                DEY
                BPL  Bs1
                RTS
BankVals        DB   $10,$11,$48,$4A,$4C,$4E
```

## OTHER TYPES

Other types I have come across are carts with an MMC type PAL. The general setup is similar to a MMC1 cart, but the PAL (labelled '109') is wired differently. Program and Character bank selection are possible, and so is setting the scrolling mode through software.

Cartridges include Gauntlet and Pac Man, both by Tengen.

This may be a special chip or a precursor to the MMC. The chip has no 'NES' label, generates MMC type signals, and seems to be used only by Tengen.

## GENERAL NES CODE EXAMPLE

Following is a listing of a fully functional, self contained NES test program. The program contains rudimentary code for background and sprite graphics, scrolling and sound. This code may be used as a base for understanding practical NES programming. It can be modified by the programmer to better understand some of the information contained in this manual.

The graphics and music for this example where created using the NES graphics and sound utilities (NICHED, NISPED and NIMCO) and including the resulting assembly files in the source.

```
                ORG     $C000

                ;
                ; NES Equates
                ;

PCR1            EQU     $2000     ; PPU Control Register 1
PCR2            EQU     $2001     ; PPU Control Register 2
PSTR            EQU     $2002     ; PPU status register
PPCR            EQU     $2003     ; PPU DMA page count
PSOR            EQU     $2005     ; PPU Scrolling Offset Register
PMAR            EQU     $2006     ; PPU Memory Address Register
PMDR            EQU     $2007     ; PPU Memory Data Register

V1R1            EQU     $4000     ; Voice 1 Register 1
V1R2            EQU     $4001     ; Voice 1 Register 2
V1R3            EQU     $4002     ; Voice 1 Register 3
V1R4            EQU     $4003     ; Voice 1 Register 4

V2R1            EQU     $4004     ; Voice 2 Register 1
V2R2            EQU     $4005     ; Voice 2 Register 2
V2R3            EQU     $4006     ; Voice 2 Register 3
V2R4            EQU     $4007     ; Voice 2 Register 4

V3R1            EQU     $4008     ; Voice 3 Register 1
V3R2            EQU     $4009     ; Voice 3 Register 2
V3R3            EQU     $400A     ; Voice 3 Register 3
V3R4            EQU     $400B     ; Voice 3 Register 4

V4R1            EQU     $400C     ; Voice 4 Register 1
V4R2            EQU     $400D     ; Voice 4 Register 2
V4R3            EQU     $400E     ; Voice 4 Register 3
V4R4            EQU     $400F     ; Voice 4 Register 4

V5R1            EQU     $4010     ; Voice 5 Register 1
```

```
V5R1          EQU    $4011       ; Voice 5 Register 2
V5R1          EQU    $4012       ; Voice 5 Register 3
V5R1          EQU    $4013       ; Voice 5 Register 4

DPNR          EQU    $4014       ; DMA Page Number Register
VMER          EQU    $4015       ; Voice Master Enable Register

JOY1PORT      EQU    $4016       ; joystick 1 port
JOY2PORT      EQU    $4017       ; joystick 2 port

SPRITEYLOC    EQU    $200        ; sprite Y loc
SPRITECHAR    EQU    $201        ; sprite character
SPRITECOLOR   EQU    $202        ; sprite color/mods
SPRITEXLOC    EQU    $203        ; sprite X location

JOYBUTA       EQU    %10000000    ; 'A' Button
JOYBUTB       EQU    %01000000    ; 'B' Button
JOYSEL        EQU    %00100000    ; 'Select'
JOYSTART      EQU    %00010000    ; 'Start'
JOYUP         EQU    %00001000    ;
JOYDOWN       EQU    %00000100    ;
JOYLEFT       EQU    %00000010    ;
JOYRIGHT      EQU    %00000001    ;

PCR1Init      EQU    $10         ; clear scrolling MSB's
                                 ; set auto increment to 1
                                 ; sprite chars at $0000 - $0FFF
                                 ; bkgnd chars at $1000 - $1FFF
                                 ; interrupt disabled
PCR2Init      EQU    $06         ; enable color
                                 ; blank sprites
                                 ; blank background

              ;
              ; Page Zero Variables
              ;

Joy1Last      EQU    $00
Joy1Data      EQU    $01
Joy1Edge      EQU    $02
Joy1Delay     EQU    $03
ScrnBlank     EQU    $04

CodeState     EQU    $0A         ; software state

ShPCR1        EQU    $10         ; PCR1 shadow register
ShPCR2        EQU    $11         ; PCR2 shadow register
VScroll       EQU    $12         ; vertical scroll offset
```

```
HScroll         EQU   $13       ; horizontal scroll offset

                ; Sprite working variables

SprPtr          EQU   $20       ; ptr to start of sprite def
SprRow          EQU   $22       ; sprite row
SprColumn       EQU   $23       ; sprite column
SprRowOfs       EQU   $24       ; row offset in pixels
SprColOfs       EQU   $25       ; column offset in pixels
CurrX           EQU   $26       ; current X coordinate
CurrY           EQU   $27       ; current Y coordinate

                ;
                ; sound variables page zero $30 - $7F
                ;

W_Attack1       EQU   $30       ; working envelope vars
W_Attack2       EQU   $31
W_Attack4       EQU   $33
W_Decay1        EQU   $34
W_Decay2        EQU   $35
W_Decay4        EQU   $37
W_Sustain1      EQU   $38
W_Sustain2      EQU   $39
W_Sustain4      EQU   $3B
W_Release1      EQU   $3C
W_Release2      EQU   $3D
W_Release4      EQU   $3F

W_Duration1     EQU   $40       ; working duration vars
W_Duration2     EQU   $41
W_Duration3     EQU   $42
W_Duration4     EQU   $43

W_SndTimer      EQU   $44
VceLoop         EQU   $45
VceInit         EQU   $47

VceAdr1         EQU   $4E
VceAdr2         EQU   $50

C_VceAdr1       EQU   $54   ; Address of current voice data
C_VceAdr2       EQU   $56
C_VceAdr3       EQU   $58
C_VceAdr4       EQU   $5A
C_FrqAdr1       EQU   $5C   ; Address of current freq table
C_FrqAdr2       EQU   $5E
C_FrqAdr3       EQU   $60
```

```
C_FrqAdr4      EQU    $62

SoundData      EQU    $64    ; pointer to new sound data
VceIndex       EQU    $66    ; current voice index
VceAdr3        EQU    $68
TempSound      EQU    $6A
RestEnable     EQU    $6B
SndTemp1       EQU    $6C
SndTemp2       EQU    $6E

V1AmpWave      EQU    $70
V1Sweep        EQU    $71
V1FreqLo       EQU    $72
V1FreqHi       EQU    $73

V2AmpWave      EQU    $74
V2Sweep        EQU    $75
V2FreqLo       EQU    $76
V2FreqHi       EQU    $77

V3Cntrl        EQU    $78
V3Dummy        EQU    $79
V3FreqLo       EQU    $7A
V3FreqHi       EQU    $7B

V4AmpWave      EQU    $7C
V4Dummy        EQU    $7D
V4FreqLo       EQU    $7E
V4FreqHi       EQU    $7F

               ; Sound variables $310 - $338

V5Rate         EQU    $0310 ; voice 5 vars
V5AdrLo        EQU    $0311
V5AdrHi        EQU    $0312
V5Size         EQU    $0313

Attack1        EQU    $0314
Attack2        EQU    $0315
Attack4        EQU    $0317
Decay1         EQU    $0318
Decay2         EQU    $0319
Decay4         EQU    $031B
Sustain1       EQU    $031C
Sustain2       EQU    $031D
Sustain4       EQU    $031F
Release1       EQU    $0320
Release2       EQU    $0321
```

```
Release4       EQU   $0323

SndTimer       EQU   $0328
VceEnable      EQU   $0329

EnvIndex       EQU   $032A
PeakVol        EQU   $032E

V1Shadow       EQU   $0334
SweepShadow    EQU   $0338

Temp           EQU   $F6     ; temporary
Ind0           EQU   $F8     ; pointer

RESVector      SEI
               LDA   #0                ; init control registers
               STA   PCR1
               STA   PCR2
Wait0          LDA   PSTR              ; wait for hardware
               BPL   Wait0             ; to stabilize
Wait1          LDA   PSTR
               BPL   Wait1
Wait2          LDA   PSTR
               BPL   Wait2
               LDX   #0
               STX   CodeState         ; clear software state
               STX   HScroll           ; clear H scroll
               STX   VScroll           ; clear V Scroll
               STX   VMER              ; quiet all sound
               DEX
               TXS                     ; init stack pointer
               LDA   #PCR1Init         ; init PCR1
               STA   ShPCR1
               STA   PCR1
               LDA   #PCR2Init         ; init PCR2
               STA   ShPCR2
               STA   PCR2
               LDA   ShPCR1            ; enable NMI interrupts
               ORA   #$80
               STA   ShPCR1
               STA   PCR1
               LDX   #SoundStrng & $FF
               LDY   #SoundStrng / 256
               JSR   SoundInit         ; do sound init

ForeGround     NOP                     ; do nothing
               NOP                     ; in the foreground
               NOP
```

```
                JMP   ForeGround

IRQVector       CLI                     ; should never happen
                RTI

NMIVector       PHP                     ; save processor registers
                PHA
                TXA
                PHA
                TYA
                PHA

                LDA   PSTR              ; clear interrupt bit
                LDA   ShPCR1            ; get PCR1 shadow
                AND   #$7F              ; disable interrupts
                STA   PCR1              ; save in register
                STA   ShPCR1            ; save in shadow

                LDA   #$00              ; Start Sprite DMA
                STA   $2003
                LDA   #$02              ; DMA page number
                STA   DPNR

                LDA   ShPCR2            ; get PCR 2 shadow
                AND   #$E7              ; blank screen/enable
                                        ; CPU access
                STA   ShPCR2
                STA   PCR2

                JSR   SetUp             ; screen memory access
                                        ; must happen here

                LDA   ShPCR2            ; get PCR 2 shadow
                ORA   #$18              ; enable screen/block
                                        ; CPU access
                STA   ShPCR2
                STA   PCR2

                JSR   ReadJoy           ; read joystick
                JSR   DoTest            ; execute main code
                JSR   SoundRefresh      ; update sound

                LDA   HScroll           ; refresh scroll registers
                STA   PSOR
                LDA   VScroll
                STA   PSOR

                LDA   ShPCR1            ; get PCR1 shadow
                ORA   #$80           ; enable interrupts
                STA   PCR1           ; save to PPU
                STA   ShPCR1         ; save to shadow

                PLA                  ; restore registers
                TAY
                PLA
                TAX
                PLA
                PLP
                RTI

                ; actual sprite display code

SetUp           LDA   CodeState
                CMP   #0
                BNE   SetUp1
                JSR   ClrChars
                JSR   ClrSprites
                JSR   ClrScreen
                INC   CodeState
                RTS

SetUp1          CMP   #1
                BNE   SetUp2
                JSR   LoadSprites
                JSR   LoadScreen
                LDA   #$A8
                STA   CurrX
                LDA   #$A0
                STA   CurrY
                INC   CodeState
SetUp2          RTS

DoTest          LDA   CodeState
                CMP   #2
                BNE   TestExit
                JSR   DoCommand
                JSR   DrawSprite
                RTS

TestExit        RTS

                ;
                ; Read controller A
                ;

ReadJoy         LDY   #1                   ; load controller 1
```

```
                STY     JOY1PORT            ; shift register
                DEY
                STY     JOY1PORT
                LDY     #8                  ; number of bits
ReadJ1          PHA                         ; save new bits
                LDA     JOY1PORT            ; get next bit(s)
                STA     Joy1Last
                LSR     A                   ; next bit of data
                ORA     Joy1Last            ; fold in first bit
                LSR     A                   ; new bit into carry
                PLA                         ; restore new bits
                ROL     A                   ; shift in new bit
                DEY                         ; dec bit counter
                BNE     ReadJ1
                LDY     Joy1Data        ; transfer previous data
                STY     Joy1Last
                STA     Joy1Data            ; save new data
                EOR     Joy1Last            ; debounce it
                AND     Joy1Data
                STA     Joy1Edge
                LDY     #$30                ; delay until repeat
                LDA     Joy1Data            ; see if new data
                CMP     Joy1Last
                BNE     ReadJ2              ; start new delay
                DEC     Joy1Delay
                BNE     ReadJ3
                STA     Joy1Edge            ; store repeat value
                LDY     #10                 ; delay between
ReadJ2          STY     Joy1Delay           ; repeats
ReadJ3          RTS

                ;
                ; process command if
                ; joystick pressed
                ;

DoCommand       LDA     Joy1Data        ; get current data
                AND     #JOYUP          ; if UP
                BEQ     DoCom1          ; no
                LDA     CurrY           ; get Y
                CMP     #$16            ; check limit
                BCC     DoCom2          ; at limit
                DEC     CurrY           ;
                DEC     CurrY           ;
                JMP     DoCom2          ;
DoCom1          LDA     Joy1Data        ; current data
                AND     #JOYDOWN        ; if DOWN
                BEQ     DoCom2          ; no
                LDA   CurrY           ; get Y
                CMP   #$B8            ; check limit
                BCS   DoCom2          ; at limit
                INC   CurrY           ;
                INC   CurrY           ;
DoCom2          LDA   Joy1Data        ; current data
                AND   #JOYLEFT        ; if LEFT
                BEQ   DoCom3          ; no
                LDA   CurrX           ; get X
                CMP   #$8             ; check limit
                BCC   DoCom4          ; at limit
                DEC   CurrX           ;
                DEC   CurrX           ;
                JMP   DoCom4          ;
DoCom3          LDA   Joy1Data        ; current data
                AND   #JOYRIGHT       ; if RIGHT
                BEQ   DoCom4          ; no
                LDA   CurrX           ; get X
                CMP   #$C8            ; check limit
                BCS   DoCom4          ; at limit
                INC   CurrX           ;
                INC   CurrX           ;
DoCom4          LDA   Joy1Data        ; current data
                AND   #JOYBUTA        ; if 'A'
                BEQ   DoCom5          ;
                LDA   HScroll         ; get horizontal
                CMP   #0              ; at limit
                BEQ   DoCom5          ; yes
                DEC   HScroll         ;
                DEC   HScroll         ;
DoCom5          LDA   Joy1Data        ; current data
                AND   #JOYBUTB        ; if 'B'
                BEQ   DoCom6          ;
                LDA   HScroll         ; get horizontal
                CMP   #$FE            ; at limit
                BEQ   Docom6          ; yes
                INC   HScroll         ;
                INC   HScroll         ;
DoCom6          LDA   Joy1Edge        ; current data
                AND   #JOYSTART       ; if 'Start'
                BEQ   DoCom7          ;
                LDA   #1              ;
                JSR   SoundStart      ; start sound
                RTS                   ;
DoCom7          LDA   Joy1Edge        ; current data
                AND   #JOYSEL         ; if 'select'
                BEQ   DoCom8          ;
                LDA   #0              ;
                JSR   SoundStart     ; stop sound
DoCom8          RTS                  ;

                ;
                ; draw the sprite
                ;

DrawSprite      LDA   #<Frame001     ; point to frame data
                STA   SprPtr         ;
                LDA   #>Frame001     ;
                STA   SprPtr+1       ;
                LDA   #0
                STA   SprRow         ; row number
                STA   SprColumn      ; column number
                STA   SprRowOfs      ; row offset
                STA   SprColOfs      ; column offset
                TAY
                TAX
Dsp10           LDA   CurrY          ; get base Y
                CLC                  ;
                ADC   SprRowOfs      ; add row offset
                STA   SPRITEYLOC,X   ; into sprite Y
                LDA   (SprPtr),Y     ; get character
                STA   SPRITECHAR,X   ; save sprite character
                INY                  ; update index
                LDA   (SprPtr),Y     ; get color
                STA   SPRITECOLOR,X  ;
                INY                  ; for next pair
                LDA   CurrX          ; get base X
                CLC                  ;
                ADC   SprColOfs      ; add column offset
                STA   SPRITEXLOC,X   ; save sprite Y
Dsp14           INX                  ; point to next
                INX                  ; physical sprite
                INX                  ;
                INX                  ;
Dsp15           LDA   SprColOfs      ; bump column offset
                CLC                  ;
                ADC   #8             ;
                STA   SprColOfs      ;
                INC   SprColumn      ;
                LDA   SprColumn      ;
                CMP   #6             ; see if at edge
                BEQ   Dsp16          ; yes
                JMP   Dsp10          ;
Dsp16           LDA   #0             ; clear column offset
                STA   SprColumn      ;
                STA   SprColOfs      ;
                LDA   SprRowOfs      ; bump row offset
                CLC                  ;
                ADC   #8             ;
                STA   SprRowOfs      ;
                INC   SprRow         ;
                LDA   SprRow         ;
                CMP   #5             ;
                BCS   Dsp17          ;
                JMP   Dsp10          ;
Dsp17           RTS                  ;


                ; This routine loads sprites
                ; from the database to their
                ; respective destinations.
                ; Blocks include sprite colors
                ; and sprite character definitions

LoadSprites     LDA   #<SpriteData   ; get starting address
                STA   Ind0           ; of data
                LDA   #>SpriteData
                STA   Ind0+1
Lspr0           LDA   PSTR           ; clear pending interrupts
                LDY   #$00
                LDA   (Ind0),Y       ; dest address low
                STA   Temp
                INY
                LDA   (Ind0),Y       ; dest address high
                STA   Temp+1
                AND   Temp           ; done if address
                CMP   #$FF           ; = $FFFF
                BEQ   LsprX
                INY
                LDA   Temp+1         ; get MSB of dest
                STA   PMAR
                LDA   Temp           ; get LSB of dest
                STA   PMAR
                LDA   #$00           ; get negative of
                SEC                  ; 16 bit data count
                SBC   (Ind0),Y
                STA   Temp
                INY
                LDA   #$00
                SBC   (Ind0),Y
                STA   Temp+1
                INY
Lspr1           LDA   (Ind0),Y       ; transfer actual data
                STA   PMDR
                INY
                BNE    Lspr2
                INC    Ind0+1          ; bump source
Lspr2           INC    Temp            ; check counter
                BNE    LSpr1           ; for more data
                INC    Temp+1
                BNE    LSpr1
                CLC                    ; update pointer
                TYA                    ; to next data block
                ADC    Ind0
                STA    Ind0
                BCC    Lspr4
                INC    Ind0+1
Lspr4           JMP    Lspr0
LsprX           RTS

SpriteData
                DW     $0000           ; sprite character set
                DW     $0200           ; total of 32 characters

                DB     $00,$FA,$22,$23,$22,$22,$22,$00
                DB     $05,$00,$08,$88,$88,$88,$88,$DD
                DB     $00,$5C,$49,$C8,$48,$49,$5C,$00
                DB     $A3,$02,$00,$02,$00,$02,$00,$A3
                DB     $00,$C1,$20,$80,$40,$20,$C1,$00
                DB     $3E,$1C,$0E,$1F,$1F,$0E,$1C,$3E
                DB     $00,$CC,$92,$88,$84,$92,$CC,$00
                DB     $33,$31,$00,$21,$31,$00,$01,$33
                DB     $00,$60,$90,$90,$F0,$90,$90,$00
                DB     $9F,$0F,$07,$07,$07,$07,$07,$6F
                DB     $00,$33,$4A,$22,$13,$4A,$32,$00
                DB     $8C,$80,$00,$80,$80,$00,$80,$CD
                DB     $00,$9C,$52,$52,$9C,$12,$12,$00
                DB     $63,$01,$00,$00,$01,$40,$C0,$ED
                DB     $00,$EF,$42,$42,$42,$42,$E2,$00
                DB     $10,$00,$10,$18,$18,$18,$08,$1D

                DB     $00,$BC,$20,$38,$20,$20,$3C,$00
                DB     $43,$01,$01,$83,$87,$83,$81,$C3
                DB     $00,$00,$00,$00,$03,$07,$07,$0F
                DB     $00,$00,$00,$00,$00,$00,$00,$00
                DB     $00,$00,$10,$38,$FC,$FF,$FF,$FF
                DB     $00,$00,$00,$00,$00,$00,$00,$00
                DB     $00,$00,$00,$00,$E7,$FF,$FF,$FF
                DB     $00,$00,$00,$00,$00,$00,$00,$00
                DB     $00,$0F,$1F,$3F,$FF,$FF,$FF,$FF
                DB     $00,$00,$00,$00,$00,$00,$00,$00
                DB     $00,$80,$C0,$E0,$F1,$F3,$F7,$FF
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $00,$00,$38,$78,$FC,$FC,$FE,$FC
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $F8,$F8,$F8,$F0,$E0,$E0,$E0,$F0
DB      $00,$00,$00,$00,$00,$00,$00,$00

DB      $F0,$FC,$FC,$FC,$FC,$FC,$F0,$F0
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $F0,$F0,$F0,$F0,$F0,$F0,$F8,$F8
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $F0,$F8,$FC,$FE,$FE,$7E,$7E,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $FF,$FF,$FF,$FF,$FF,$00,$00,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $FF,$FF,$FF,$7F,$7F,$7C,$3C,$18
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $FF,$FF,$FF,$F8,$F0,$E0,$00,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $FF,$FF,$FF,$F7,$F3,$F1,$F0,$E0
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $1F,$1F,$1F,$0F,$07,$03,$01,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00

DB      $1F,$1F,$1F,$1F,$03,$07,$0F,$1F
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $3F,$1F,$0F,$1F,$3F,$1F,$0F,$1F
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $1F,$1F,$0F,$07,$07,$07,$0F,$1F
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $FF,$FE,$FE,$FE,$FE,$FE,$FE,$FF
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00

DW      $3F10           ; colors
DW      $0010
DB      $0F,$10,$07,$1C
DB      $0F,$07,$04,$08
DB      $0F,$00,$2D,$11
DB      $0F,$29,$2C,$26

DB      $FF,$FF
```

```
                DB    $FF,$FF

                ;
                ; Frame Data [Sprite#,Sprite Color/Flip]
                ;

Frame001        DB    $09,$01,$0A,$01,$0B,$01
                DB    $0C,$01,$0D,$01,$0E,$01
                DB    $1A,$01,$00,$00,$01,$00
                DB    $02,$00,$03,$00,$0F,$01
                DB    $19,$01,$1B,$00,$1C,$00
                DB    $04,$00,$1B,$00,$10,$01
                DB    $18,$01,$05,$00,$06,$00
                DB    $07,$00,$08,$00,$11,$01
                DB    $17,$01,$16,$01,$15,$01
                DB    $14,$01,$13,$01,$12,$01
                DB    $FF,$00,$FF,$00,$FF,$00
                BD    $FF,$00,$FF,$00,$FF,$00

                ; This routine will transfer blocks
                ; from the database to their
                ; respective destinations.
                ; Blocks include Background screen,
                ; color and character definitions

LoadScreen      LDA   #<ScreenData   ; get starting address
                STA   Ind0           ; of data
                LDA   #>ScreenData
                STA   Ind0+1
Lscr0           LDA   PSTR           ; clear pending interrupts
                LDY   #$00
                LDA   (Ind0),Y       ; dest address low
                STA   Temp
                INY
                LDA   (Ind0),Y       ; dest address high
                STA   Temp+1
                AND   Temp           ; done if address
                CMP   #$FF           ; = $FFFF
                BEQ   LscrX
                INY
                LDA   Temp+1         ; get MSB of dest
                STA   PMAR
                LDA   Temp           ; get LSB of dest
                STA   PMAR
                LDA   #$00           ; get negative of
                SEC                  ; 16 bit data count
                SBC   (Ind0),Y
                STA   Temp
                INY
                LDA     #$00
                SBC     (Ind0),Y
                STA     Temp+1
                INY
Lscr1           LDA     (Ind0),Y        ; transfer actual data
                STA     PMDR
                INY
                BNE     Lscr2
                INC     Ind0+1          ; bump source
Lscr2           INC     Temp            ; check counter
                BNE     LScr1           ; for more data
                INC     Temp+1
                BNE     LScr1
                CLC                     ; update pointer
                TYA                     ; to next data block
                ADC     Ind0
                STA     Ind0
                BCC     Lscr4
                INC     Ind0+1
Lscr4           JMP     Lscr0
LscrX           RTS

ScreenData
                DW      $1000           ; character set
                DW      $0300

                DB      $38,$4C,$C6,$C6,$C6,$64,$38,$00
                DB      $00,$00,$00,$00,$00,$00,$00,$00
                DB      $18,$38,$18,$18,$18,$18,$7E,$00
                DB      $00,$00,$00,$00,$00,$00,$00,$00
                DB      $7C,$C6,$0E,$3C,$78,$E0,$FE,$00
                DB      $00,$00,$00,$00,$00,$00,$00,$00
                DB      $7E,$0C,$18,$3C,$06,$C6,$7C,$00
                DB      $00,$00,$00,$00,$00,$00,$00,$00
                DB      $1C,$3C,$6C,$CC,$FE,$0C,$0C,$00
                DB      $00,$00,$00,$00,$00,$00,$00,$00
                DB      $FC,$C0,$FC,$06,$06,$C6,$7C,$00
                DB      $00,$00,$00,$00,$00,$00,$00,$00
                DB      $3C,$60,$C0,$FC,$C6,$C6,$7C,$00
                DB      $00,$00,$00,$00,$00,$00,$00,$00
                DB      $FE,$C6,$0C,$18,$30,$30,$30,$00
                DB      $00,$00,$00,$00,$00,$00,$00,$00

                DB      $7C,$C6,$C6,$7C,$C6,$C6,$7C,$00
                DB      $00,$00,$00,$00,$00,$00,$00,$00
                DB      $7C,$C6,$C6,$7E,$06,$0C,$78,$00
                DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $38,$6C,$C6,$C6,$FE,$C6,$C6,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $FC,$C6,$C6,$FC,$C6,$C6,$FC,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $3C,$66,$C0,$C0,$C0,$66,$3C,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $F8,$CC,$C6,$C6,$C6,$CC,$F8,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $FE,$C0,$C0,$FC,$C0,$C0,$FE,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $FE,$C0,$C0,$FC,$C0,$C0,$C0,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00

DB      $3E,$60,$C0,$CE,$C6,$66,$3E,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $C6,$C6,$C6,$FE,$C6,$C6,$C6,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $7E,$18,$18,$18,$18,$18,$7E,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $1E,$06,$06,$06,$C6,$C6,$7C,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $C6,$CC,$D8,$F0,$F8,$DC,$CE,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $60,$60,$60,$60,$60,$60,$7E,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $C6,$EE,$FE,$FE,$D6,$C6,$C6,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $C6,$E6,$F6,$FE,$DE,$CE,$C6,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00

DB      $7C,$C6,$C6,$C6,$C6,$C6,$7C,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $FC,$C6,$C6,$C6,$FC,$C0,$C0,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $7C,$C6,$C6,$C6,$DE,$CC,$7A,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $FC,$C6,$C6,$CE,$F8,$DC,$CE,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $78,$CC,$C0,$7C,$06,$C6,$7C,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $7E,$18,$18,$18,$18,$18,$18,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $C6,$C6,$C6,$C6,$C6,$C6,$7C,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $C6,$C6,$C6,$EE,$7C,$38,$10,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00

DB      $C6,$C6,$D6,$FE,$FE,$EE,$C6,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $C6,$EE,$7C,$38,$7C,$EE,$C6,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $66,$66,$66,$3C,$18,$18,$18,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $FE,$0E,$1C,$38,$70,$E0,$FE,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $18,$18,$18,$00,$00,$00,$00,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $66,$FF,$FF $66,$66,$FF,$FF,$66
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00

DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $FE,$FE,$06,$06,$E6,$E6,$66,$66
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $00,$00,$FF,$FF,$00,$00,$FF,$FF
DB      $7F,$7F,$60,$60,$67,$67,$66,$66
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $66,$66,$66,$66,$66,$66,$66,$66
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $66,$66,$E6,$E6,$06,$06,$FE,$FE
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$00,$00,$FF,$FF,$00,$00
DB      $66,$66,$67,$67,$60,$60,$7F,$7F
DB      $00,$00,$00,$00,$00,$00,$00,$00

DW      $3F00           ; colors
DW      $0010
DB      $0F,$28,$08,$0F
DB      $0F,$11,$19,$0F
DB      $0F,$00,$2D,$0F
DB      $0F,$26,$2C,$0F

DW      $2000   ; character map screen 1
DW      $400
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $9B,$9B,$2B,$2A,$2A,$2A,$2A,$2A
DB      $2A,$2A,$2A,$2A,$2A,$2A,$2A,$2A
DB      $2A,$2A,$2A,$2A,$2A,$2A,$2A,$2A
DB      $2A,$2A,$2A,$2A,$2A,$29,$CA,$CA
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$17,$12,$17
DB      $1D,$0E,$17,$0D,$18,$FF,$0E,$21
DB      $0A,$16,$19,$15,$0E,$FF,$0C,$18
DB      $0D,$0E,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$D0,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$0B,$22,$FF
DB      $1C,$0C,$1E,$15,$19,$1D,$1E,$1B
DB      $0E,$0D,$FF,$1C,$18,$0F,$1D,$20
DB      $0A,$1B,$0E,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $D6,$D6,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$13,$18,$22
DB      $1C,$1D,$12,$0C,$14,$FF,$1D,$18
DB      $FF,$16,$18,$1F,$0E,$FF,$1C,$19
DB      $1B,$12,$1D,$0E,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$24
DB      $0A,$24,$FF,$1D,$18,$FF,$1C,$0C
DB      $1B,$18,$15,$15,$FF,$15,$0E,$0F
DB      $1D,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$24
DB      $0B,$24,$FF,$1D,$18,$FF,$1C,$0C
DB      $1B,$18,$15,$15,$FF,$1B,$12,$10
DB      $11,$1D,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$A6,$A6,$A6,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$24
DB      $1C,$1D,$0A,$1B,$1D,$24,$A6,$0F
DB      $18,$1B,$FF,$16,$1E,$1C,$12,$0C
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$24,$1C
DB      $0E,$15,$0E,$0C,$1D,$24,$FF,$0F
DB      $18,$1B,$FF,$1C,$12,$15,$0E,$17
DB      $0C,$0E,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$1C,$0C,$1B,$0E,$0E
DB      $17,$FF,$25,$01,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2F,$2E,$2E,$2E,$2E,$2E
DB      $2E,$2E,$2E,$2E,$2E,$2E,$2E,$2E
DB      $2E,$2E,$2E,$2E,$2E,$2E,$2E,$2E
DB      $2E,$2E,$2E,$2E,$2E,$2D,$FF,$FF
DB      $FF,$FF,$D0,$D0,$D0,$D0,$D0,$D0
DB      $D0,$D0,$D0,$D0,$D0,$D0,$D0,$D0
DB      $D0,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$D0,$FF,$FF
DB      $FF,$FF,$D0,$D0,$D0,$D0,$D0,$D0
DB      $D0,$D0,$D0,$D0,$D0,$D0,$D0,$D0
DB      $D0,$D0,$D0,$D0,$D0,$D0,$D0,$D0
DB      $D0,$D0,$D0,$D0,$D0,$D0,$FF,$FF


DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $00,$55,$55,$99,$AA,$EE,$33,$00
DB      $00,$AA,$AF,$AF,$A7,$A5,$A5,$00
DB      $00,$42,$50,$50,$50,$50,$18,$00
DB      $00,$8C,$AF,$AF,$AF,$AF,$21,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00
DB      $00,$00,$00,$00,$00,$00,$00,$00

DW      $2400           ; character map
DW      $400
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $9B,$9B,$2B,$2A,$2A,$2A,$2A,$2A
DB      $2A,$2A,$2A,$2A,$2A,$2A,$2A,$2A
DB      $2A,$2A,$2A,$2A,$2A,$2A,$2A,$2A
DB      $2A,$2A,$2A,$2A,$2A,$29,$CA,$CA
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$17,$12,$17
DB      $1D,$0E,$17,$0D,$18,$FF,$0E,$21
DB      $0A,$16,$19,$15,$0E,$FF,$0C,$18
DB      $0D,$0E,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$D0,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$0B,$22,$FF
DB      $1C,$0C,$1E,$15,$19,$1D,$1E,$1B
DB      $0E,$0D,$FF,$1C,$18,$0F,$1D,$20
DB      $0A,$1B,$0E,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $D6,$D6,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$13,$18,$22
DB      $1C,$1D,$12,$0C,$14,$FF,$1D,$18
DB      $FF,$16,$18,$1F,$0E,$FF,$1C,$19
DB      $1B,$12,$1D,$0E,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$24
DB      $0A,$24,$FF,$1D,$18,$FF,$1C,$0C
DB      $1B,$18,$15,$15,$FF,$15,$0E,$0F
DB      $1D,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$24
DB      $0B,$24,$FF,$1D,$18,$FF,$1C,$0C
DB      $1B,$18,$15,$15,$FF,$1B,$12,$10
DB      $11,$1D,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$A6,$A6,$A6,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$24
DB      $1C,$1D,$0A,$1B,$1D,$24,$A6,$0F
DB      $18,$1B,$FF,$16,$1E,$1C,$12,$0C
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$24,$1C
DB      $0E,$15,$0E,$0C,$1D,$24,$FF,$0F
DB      $18,$1B,$FF,$1C,$12,$15,$0E,$17
DB      $0C,$0E,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2C,$FF,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$1C,$0C,$1B,$0E,$0E
DB      $17,$FF,$25,$02,$FF,$FF,$FF,$FF
DB      $FF,$FF,$FF,$FF,$FF,$2C,$FF,$FF
DB      $FF,$FF,$2F,$2E,$2E,$2E,$2E,$2E
DB      $2E,$2E,$2E,$2E,$2E,$2E,$2E,$2E
DB      $2E,$2E,$2E,$2E,$2E,$2E,$2E,$2E
DB      $2E,$2E,$2E,$2E,$2E,$2D,$FF,$FF
DB      $FF,$FF,$D0,$D0,$D0,$D0,$D0,$D0
```

```
                DB     $D0,$D0,$D0,$D0,$D0,$D0,$D0,$D0
                DB     $D0,$FF,$FF,$FF,$FF,$FF,$FF,$FF
                DB     $FF,$FF,$FF,$FF,$FF,$D0,$FF,$FF
                DB     $FF,$FF,$D0,$D0,$D0,$D0,$D0,$D0
                DB     $D0,$D0,$D0,$D0,$D0,$D0,$D0,$D0
                DB     $D0,$D0,$D0,$D0,$D0,$D0,$D0,$D0
                DB     $D0,$D0,$D0,$D0,$D0,$D0,$FF,$FF


                DB     $40,$50,$50,$50,$50,$50,$50,$10
                DB     $44,$5F,$5F,$97,$A5,$E9,$32,$11
                DB     $44,$00,$05,$05,$09,$0A,$0A,$11
                DB     $44,$86,$A5,$A5,$A5,$A5,$29,$11
                DB     $44,$C0,$F0,$F0,$F0,$F0,$31,$11
                DB     $44,$00,$00,$00,$00,$00,$00,$11
                DB     $44,$50,$50,$50,$50,$50,$50,$11
                DB     $00,$00,$00,$00,$00,$00,$00,$00

                DB     $FF,$FF

                ;
                ; clear the screen
                ;

ClrScreen       LDA    #$20              ; dest MSB = $20
                STA    PMAR
                LDA    #0                ; dest LSB = $00
                STA    PMAR
                LDX    #$00
                LDY    #$00
                LDA    #$FF
ClrScreen0      STA    PMDR              ; store character
                DEX                      ; $400 times
                BNE    ClrScreen0
                INY
                CPY    #$04
                BNE    ClrScreen0
                RTS

                ; clear the background
                ; character set area

ClrChars        LDA    #$10
                STA    PMAR
                LDA    #$00
                STA    PMAR
                LDA    #0
                TAX
                TAY
ClrCh           STA    PMDR
                INX
                BNE    ClrCh
                LDA    PSTR
                INY
                CPY    #$10
                BNE    ClrCh
                RTS

                ; clear the sprite
                ; character area

ClrSprites      LDA    #$00             ; set up sprite
                STA    PMAR             ; character
                LDA    #$00             ; address
                STA    PMAR             ;
                LDA    #0               ; clear indices
                TAX                     ;
                TAY                     ;
ClrSp           STA    PMDR             ; store a 0
                INX                     ;
                BNE    ClrSp            ;
                INY                     ;
                CPY    #$10             ;
                BNE    ClrSp            ;
                LDX    #0               ;
                TXA                     ;
ClrSp1          STA    SPRITEYLOC,X     ; make sprite
                INX                     ; disappear
                INX                     ;
                INX                     ;
                INX                     ;
                BNE    ClrSp1           ;
                RTS                     ;

                ;
                ; Sound Driver
                ;

SndInit         JMP    SoundInit
SndRefresh      JMP    SoundRefresh
SndStart        JMP    SoundStart

                ; Init all sound registers
                ; Pass address of sound table in X,Y

SoundInit       LDA    #$0F             ; enable voices
```

```
                STA    VMER
                LDA    #$C0
                STA    JOY2PORT
                LDA    #$F0
                STA    V1R1
                STA    V3R2
                STA    V4R1
                LDA    #$80
                STA    V3R1
                LDA    #$78
                STA    V1R2
                STA    V2R2
                STA    V3R2
                STA    V4R2
                LDA    #$F0
                STA    V1R3
                STA    V2R3
                STA    V3R3
                STA    V4R3
                LDA    #$08
                STA    V1R4
                STA    V2R4
                STA    V3R4
                STA    V4R4
                LDA    #$00
                STA    RestEnable
                STA    W_SndTimer
                STA    SndTimer
                STA    VceEnable
                STA    VceLoop
                STA    VceLoop+1
                STA    VceLoop+2
                STA    VceLoop+3
                STA    VceInit
                STA    VceInit+1
                STA    VceInit+2
                STA    VceInit+3
                STX    SoundData
                STY    SoundData+1
                RTS

                ; Load a single voice with data

LoadVoice       LDX    VceIndex
                LDA    #$01
                STA    VceInit,X
                RTS
```

```
LoadInit        LDA     #$00
                STA     VceInit,X
                LDA     V1Sweep,Y
                STA     V1R2,Y
                LDA     V1FreqLo,Y
                STA     V1R3,Y
                LDA     V1FreqHi,Y
                STA     V1R4,Y
                JMP     LDC2

                ; load voice controller register

LoadCntrl       LDX     #$00
LDC1            LDA     BitTest,X
                AND     VceEnable
                BEQ     LDC2
                AND     RestEnable
                BNE     LDC2
                LDY     INTAB,X
                LDA     V1AmpWave,Y
                STA     V1R1,Y
                LDA     VceInit,X
                BNE     LoadInit
LDC2            INX
                CPX     #$04
                BCC     LDC1
                RTS

BitTest         DB      $01,$02,$04,$08,$10
INTAB           DB      $00,$04,$08,$0C,$10

                ; Start a new sound

SoundStart      ASL     A
                TAY
                LDA     (SoundData),Y
                STA     VceAdr1
                INY
                LDA     (SoundData),Y
                STA     VceAdr1+1
SNDST1          LDY     #0
                LDA     (VceAdr1),Y
                BMI     SndStx
                AND     #$03
                STA     VceIndex
                TAX
                LDA     #$00
                STA     EnvIndex,X
                LDA     BitTest,X
                ORA     VceEnable
                STA     VceEnable
                TXA
                ASL     A
                TAX
                INY
                LDA     (VceAdr1),Y
                STA     C_VceAdr1,X
                INY
                LDA     (VceAdr1),Y
                STA     C_VceAdr1+1,X
                INY
                LDA     (VceAdr1),Y
                STA     C_FrqAdr1,X
                INY
                LDA     (VceAdr1),Y
                STA     C_FrqAdr1+1,X
                JSR     GetData
                LDA     #5
                CLC
                ADC     VceAdr1
                STA     VceAdr1
                BCC     SNDST2
                INC     VceAdr1+1
SNDST2          Jmp     SndSt1
SndStx          RTS

                ; Get Sound Data

GetData         LDA     VceIndex
                ASL     A
                TAX
                LDA     C_VceAdr1,X
                STA     VceAdr2
                LDA     C_VceAdr1+1,X
                STA     VceAdr2+1
GData           LDY     #$00
                LDA     (VceAdr2),Y
                BMI     DoCmnd
                JSR     StartNote
                JMP     NewAdr

                ; Process command

DoCmnd          AND     #$70
                LSR     A
                LSR     A
                LSR   A
                LSR   A
                CMP   #7
                BEQ   DCMD9
                PHA
                ASL   A
                TAX
                LDA   CommandTab,X
                STA   VceAdr3
                LDA   CommandTab+1,X
                STA   VceAdr3+1
                JSR   CmdJump
                PLA
                CMP   #3
                BEQ   NewAdr
                JMP   GData

                ; Store new data address

NewAdr          LDA   VceIndex
                ASL   A
                TAX
                LDA   VceAdr2
                STA   C_VceAdr1,X
                LDA   VceAdr2+1
                STA   C_VceAdr1+1,X
DCMD9           RTS

CmdJump         JMP   (VceAdr3)         ; jump to command

CommandTab      DW    SET_ENVL          ; pass envelope data
                DW    SET_CLICKS        ; pass jiffies/count
                DW    QUIT_VOICE        ; done with voice
                DW    REST_VOICE        ; get duration of rest
                DW    LOOP_VOICE        ; loop destination
                                        ; and count
                DW    Dummy
                DW    Dummy
                DW    Dummy
                DW    Dummy

Dummy           RTS

LOOP_VOICE      INY
                LDA   (VceAdr2),Y
                STA   SndTemp2
                INY
                LDA   (VceAdr2),Y
                STA    SndTemp2+1
                INY
                LDX    VceIndex
                LDA    VceLoop,X
                BNE    CNTON
                LDA    (VceAdr2),Y
                STA    VceLoop,X
CNTON           DEC    VceLoop,X
                BEQ    ENDLOOP
LpAdr           LDA    SndTemp2
                STA    VceAdr2
                LDA    SndTemp2+1
                STA    VceAdr2+1
                RTS
ENDLOOP         LDA    #4
                JSR    ADDVceAdr2
                RTS

StartNote       LDX    VceIndex
                PHA
                LDA    BitTest,X
                EOR    #$FF
                AND    RestEnable
                STA    RestEnable
                PLA
                AND    #$1F
                ASL    A
                STA    W_Duration1,X
                TXA
                ASL    A
                TAX
                LDA    C_FrqAdr1,X
                STA    VceAdr3
                LDA    C_FrqAdr1+1,X
                STA    VceAdr3+1
                TXA
                ASL    A
                TAX
                CPX    #12
                BEQ    DOVC3
                INY
                LDA    (VceAdr2),Y
                ASL    A
                TAY
                LDA    (VceAdr3),Y
                STA    V1FreqLo,X
                INY
                LDA    (VceAdr3),Y
                STA    V1FreqHi,X

STBACK          LDX    VceIndex
                CPX    #2
                BEQ    STNOTE9
                LDA    Attack1,X
                STA    W_Attack1,X
                LDA    Decay1,X
                STA    W_Decay1,X
                LDA    Sustain1,X
                STA    W_Sustain1,X
                LDA    Release1,X
                STA    W_Release1,X
STNOTE9         TXA
                ASL    A
                ASL    A
                TAY
                LDA    V1Shadow,X
                STA    V1AmpWave,Y
                LDA    SweepShadow,X
                STA    V1Sweep,Y
                LDA    #$00
                STA    EnvIndex,X
                LDA    #2
                JSR    ADDVceAdr2
                JSR    LoadVoice
                RTS

DOVC3           INY
                LDA    (VceAdr2),Y
                STA    V1FreqLo,X
                LDA    #$00
                STA    V1FreqHi,X
                JMP    STBACK

SET_CLICKS      INY
                LDA    (VceAdr2),Y
                STA    SndTimer
                STA    W_SndTimer
                LDA    #2
                JSR    ADDVceAdr2
                RTS

QUIT_VOICE      LDA    #$01
                JSR    ADDVceAdr2
                LDX    VceIndex
                LDA    BitTest,X
                EOR    #$FF
                AND     VceEnable
                STA     VceEnable

QTVOICE         LDA     INTAB,X
                TAX
                CPX     #8
                BEQ     QT2
                LDA     V1AmpWave,X
                AND     #$F0
                STA     V1AmpWave,X
                STA     V1R1,X
                RTS

QT2             LDA     #$80
                STA     V1AmpWave,X
                STA     V1R1,X
                RTS

REST_VOICE      LDX     VceIndex
                JSR     QTVOICE
                LDX     VceIndex
                LDA     BitTest,X
                ORA     RestEnable
                STA     RestEnable
                LDA     (VceAdr2),Y
                AND     #$0F
                STA     W_Duration1,X
                LDA     #1
                JSR     ADDVceAdr2
                RTS

SET_ENVL        LDA     VceIndex
                CMP     #2
                BEQ     STENV9
                INY
                STY     SndTemp1
                ASL     A
                ASL     A
                TAX
                LDA     (VceAdr2),Y
                AND     #$03
                TAY
                LDA     WVE,Y
                STA     V1AmpWave,X
                LDY     SndTemp1
                INY
                LDA     (VceAdr2),Y
                AND     #$0F
                TAX
                LDA     TempSound
                STA     V1AmpWave,X
                JMP     SndRefresh3

MNJMP           JMP     (VceAdr3)

ENVTAB          DW      DO_Attack
                DW      DO_Decay
                DW      DO_Sustain
                DW      DO_Release

DO_Release      LDA     TempSound
                AND     #$0F
                BEQ     DOR9
                SEC
                SBC     Release1,X
                BCS     DOR1
                LDA     #$00
DOR1            STA     SndTemp1
                LDA     TempSound
                AND     #$F0
                ORA     SndTemp1
                STA     TempSound
DOR9            RTS

DO_Attack       LDA     TempSound
                AND     #$0F
                CMP     PeakVol,X
                BEQ     ATK9
                CLC
                ADC     W_Attack1,X
                CMP     PeakVol,X
                BCC     ATK1
                LDA     PeakVol,X
ATK1            STA     SndTemp1
                LDA     TempSound
                AND     #$F0
                ORA     SndTemp1
                STA     TempSound
                RTS
ATK9            LDA     #$01
                STA     EnvIndex,X
                RTS

DO_Decay        LDA     W_Decay1,X
                BEQ     DOD2
                LDA     TempSound
                AND     #$0F
                BEQ     DOD9
                DEC     TempSound
DOD9            DEC     W_Decay1,X
                RTS

DOD2            LDA     #2
                STA     EnvIndex,X
                RTS

DO_Sustain      DEC     W_Sustain1,X
                BPL     DOS1
                RTS
DOS1            LDA     #3
                STA     EnvIndex,X
                RTS

                ; Pointers To Sound Data

SoundStrng      DW      Snd0,Snd1

                ; Sound Setup Tables

Snd0            DB      0
                DW      SoundOff
                DW      NoteData
                DB      1
                DW      SoundOff
                DW      NoteData
                DB      2
                DW      SoundOff
                DW      NoteData
                DB      3
                DW      SoundOff
                DW      NoteData
                DB      $FF

Snd1            DB      0
                DW      VC1
                DW      NoteData+24
                DB      1
                DW      LIKE_HAL
                DW      NoteData+24
                DB      $FF
                ORA     V1AmpWave,X
                STA     V1AmpWave,X
                INY
                LDA     (VceAdr2),Y
                BNE     STENV2
                LDA     #$07
                JMP     STENV1
STENV2          SEC
                SBC     #1
                AND     #$0F
                ORA     #$80
STENV1          STA     V1Sweep,X
                LDX     VceIndex
                INY
                LDA     (VceAdr2),Y
                STA     Attack1,X
                INY
                LDA     (VceAdr2),Y
                STA     Decay1,X
                INY
                LDA     (VceAdr2),Y
                STA     Sustain1,X
                INY
                LDA     (VceAdr2),Y
                STA     Release1,X
                INY
                LDA     (VceAdr2),Y
                STA     PeakVol,X
STENV9          LDA     #$FF
                STA     V3Cntrl
                LDA     VceIndex
                TAY
                ASL     A
                ASL     A
                TAX
                LDA     V1AmpWave,X
                STA     V1Shadow,Y
                LDA     V1Sweep,X
                STA     SweepShadow,Y
                LDA     #9
ADDVceAdr2      CLC
                ADC     VceAdr2
                STA     VceAdr2
                BCC     STENV9J
                INC     VceAdr2+1
STENV9J         RTS

WVE             DB      $30,$70,$B0,$F0
```

```
                ; Main sound refresh routine

SoundRefresh    DEC     W_SndTimer
                LDA     W_SndTimer
                EOR     #$FF
                BNE     SndRefresh9
                LDA     SndTimer
                STA     W_SndTimer
                LDA     #$00
                STA     VceIndex
SndRefresh1     TAX
                LDA     VceEnable
                AND     BitTest,X
                BNE     SndRefresh31
                LDA     RestEnable
                AND     BitTest,X
                BEQ     SndRefresh3
SndRefresh31    DEC     W_Duration1,X
                BNE     SndRefresh2
                JSR     GetData
SndRefresh3     INC     VceIndex
                LDA     VceIndex
                CMP     #4
                BCC     SndRefresh1
                JSR     LoadCntrl
SndRefresh9     RTS

SndRefresh2     LDX     VceIndex
                CPX     #2
                BEQ     SndRefresh3
                LDA     EnvIndex,X
                ASL     A
                TAX
                LDA     ENVTAB,X
                STA     VceAdr3
                LDA     ENVTAB+1,X
                STA     VceAdr3+1
                LDA     VceIndex
                TAX
                ASL     A
                ASL     A
                TAY
                PHA
                LDA     V1AmpWave,Y
                STA     TempSound
                JSR     MNJMP
                PLA
                ; Voice and Command Data.
                ;
                ; If bit 7 is set the byte is a command byte.
                ; $80 = Set Envelope followed by 8
                ; description bytes.
                ;   Duty Cycle  = 1..3
                ;   Amplitude   = 1..15 starting amplitude
                ;   Sweep       = 0 no sweep, 1..8 down,
                ;                           9..16 up
                ;   Attack      = 0..15 rate of attack
                ;   Decay       = 0..15 rate of decay
                ;   Sustain     = 0..31 length of sustain
                ;                       after decay
                ;   Release     = 0..15 rate of release
                ;   Peak Volume = 0..15
                ; $90 = Set clicks/number of jiffies per
                ;   duration count.
                ; $A0 = Quit voice, followed by $FF.
                ; $B0 = Rest. Rest duration is specified
                ;   by lower nibble.
                ; $C0 = Loop Voice. Causes a loop back to a
                ;   specified label. Must be followed by a two
                ;   byte loop label and a single byte loop
                ;    count.
                ;
                ; If bit 7 is reset then data is actual note
                ;  data.
                ; For voices 1..3:
                ;   First byte  = duration 0..15
                ;   Second byte = pointer to frequencies.
                ;     There may be
                ;     256 frequencies in a frequency table.
                ; For voice 4:
                ;   First byte  = duration 0..15
                ;   Second byte = frequency (lower nibble).
                ;     Bit 7 specifies
                ;     random (b7=0) or periodic (b7=1).
                ;

SoundOff        DB    $A0,$FF

VC1             DB    $90,$01
                DB    $80,$01,$0F,$00,$03,$02,$20,$01,$0F
                DB    $10,$0D,$08,$11,$08,$14,$0C,$0C
                DB    $02,$0D,$02,$0F,$10,$0D
                DB    $A0,$FF

LIKE_HAL        DB    $80,$01,$0F,$00,$03,$02,$20,$01,$0F
```

```
PLOOP           DB      $04,$01,$04,$08,$04,$05,$04,$08
                DB      $C0
                DW      PLOOP
                DB      $02
                DB      $04,$03,$04,$08,$04,$06,$04,$08
                DB      $04,$01,$04,$08,$04,$05,$04,$08
                DB      $A0,$FF

VC2             DB      $80,$03,$0F,$00,$03,$02,$20,$01,$0F
                DB      $02,$11,$04,$0C,$03,$0F,$04,$0E
                DB      $02,$11,$04,$0C,$03,$0F,$0F,$0E
                DB      $A0,$FF

BASSO           DB      $80,$03,$02,$00,$02,$04,$20,$01,$09
                DB      $90,$02
                DB      $04,$09,$07,$15,$B1,$04,$15
                DB      $C0
                DW      BASSO
                DB      $04
                DB      $A0,$FF

HARMONY         DB      $80,$03,$0F,$00,$03,$02,$20,$01,$08
                DB      $08,$00,$06,$02,$02,$04,$04,$02,$0C,$00
                DB      $A0,$FF

MELODY          DB      $80,$03,$0F,$00,$03,$02,$20,$01,$0F
                DB      $08,$04,$06,$05,$02,$07,$04,$05,$0C,$04
                DB      $A0,$FF

VC3             DB      $80,$02,$0F,$00,$03,$02,$20,$01,$0F
                DB      $06,$05
                DB      $07,$03
                DB      $02,$05,$04,$00
                DB      $03,$03,$0F,$02
                DB      $A0,$FF

VC4             DB      $80,$02,$0F,$00,$03,$02,$20,$05,$0F
                DB      $02,$F6,$04,$04,$03,$F7,$04,$07
                DB      $02,$11,$04,$0C,$03,$0F,$0F,$0E
                DB      $A0,$FF
VC5             DB      $90,$00
                DB      $80,$01,$0F,$00,$03,$02,$20,$01,$0F
                DB      $02,$05,$04,$00,$03,$03,$04,$02
LP5             DB      $02,$11,$04,$0C,$03,$0F,$04,$0E
                DB      $C0
                DW      LP5
                DB      $07
                DB      $A0,$FF

                ; Table of frequency values.

NOTEDATA        DB      $AD,$06,$4A,$06,$EF,$05
                DB      $9B,$05,$4B,$05,$FE,$04
                DB      $B5,$04,$73,$04,$30,$04
                DB      $F5,$03,$BD,$03,$87,$03
                DB      $56,$03,$25,$03,$F8,$02
                DB      $CD,$02,$A5,$02,$7F,$02
                DB      $5B,$02,$3A,$02,$19,$02
                DB      $FB,$01,$DE,$01,$C3,$01
                DB      $AA,$01,$92,$01,$7B,$01
                DB      $66,$01,$52,$01,$3F,$01
                DB      $2D,$01,$1C,$01,$0C,$01
                DB      $FD,$00,$EF,$00,$E1,$00
                DB      $D4,$00,$C9,$00,$BD,$00
                DB      $B3,$00,$A9,$00,$9F,$00
                DB      $96,$00,$8E,$00,$86,$00
                DB      $7E,$00,$77,$00,$70,$00
                DB      $6A,$00,$64,$00,$5E,$00
                DB      $59,$00,$54,$00,$4F,$00
                DB      $4A,$00,$47,$00,$42,$00
                DB      $3E,$00,$3B,$00,$38,$00
                DB      $35,$00,$31,$00,$2E,$00
                DB      $2C,$00,$29,$00,$27,$00
                DB      $25,$00,$23,$00,$21,$00
                DB      $1F,$00,$1D,$00,$1B,$00

                ORG     $FFFA

                DW      NMIVector       ; NMI service routine
                DW      RESVector       ; Reset entry routine
                DW      IRQVector       ; IRQ service routine

                END
```

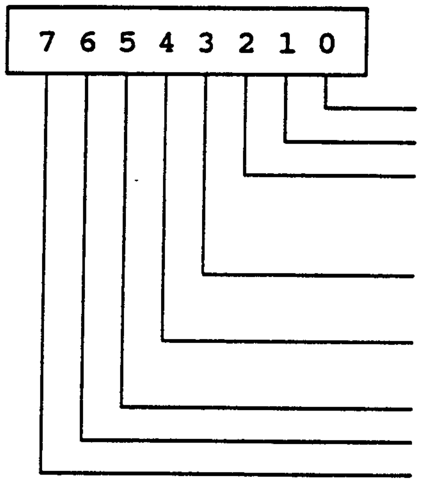
## PPU MEMORY MAP

|  |  |
|---|---|
| $0000 - $1FFF | Sprite and background character set definitions. The sprite/background sets may be assigned to either the upper or lower 4K area under program control. Both sprites and background characters may use the same area. (Memory on cartridge). |
| $2000 - $23BF | Screen 1 Character Memory |
| $23C0 - $23FF | Screen 1 Color Memory |
| $2400 - $27BF | Screen 2 Character Memory |
| $27C0 - $27FF | Screen 2 Color Memory |
| $2000 - $23BF | Screen 3 Character Memory (on cartridge) |
| $23C0 - $23FF | Screen 3 Color Memory |
| $2400 - $27BF | Screen 4 Character Memory (on cartridge) |
| $27C0 - $27FF | Screen 4 Color Memory |

## CPU MEMORY MAP

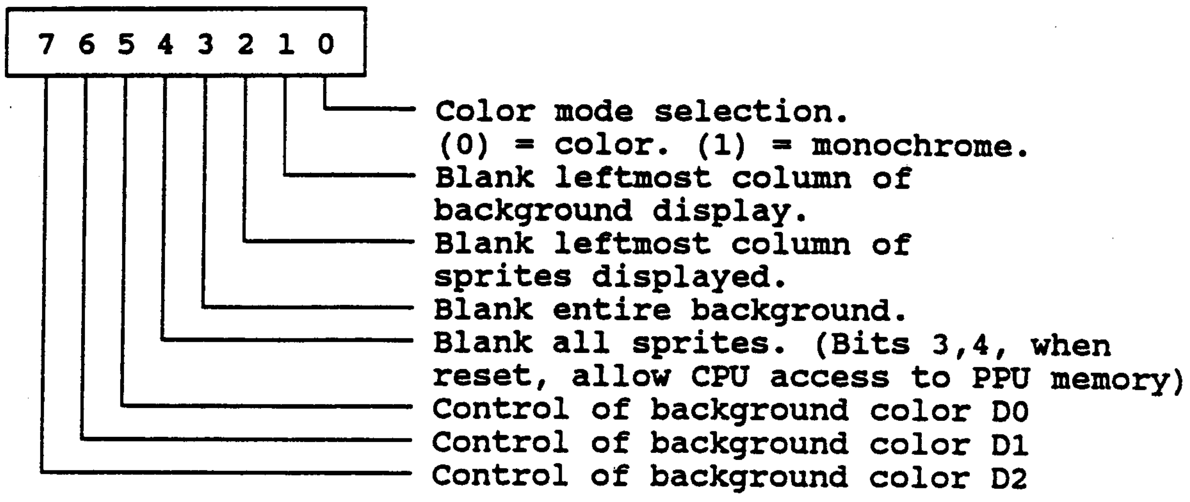
|  |  |
|---|---|
| $0000 - $00FF | Page Zero Memory. |
| $0100 - $01FF | 6502 Stack and General Purpose Memory |
| $0200 - $02FF | Sprite Definition page |
| $0300 - $07FF | General Purpose Memory |

$2000 PCR1 PPU control register 1.



MSB of horizontal scroll offset.
MSB of vertical scroll offset.
Memory transfer auto increment mode. (0) = increment by 1. (1) = increment by 32.
Sprite character memory area start. (0) = $0000. (1) = $1000.
Background character memory area start. (0) = $0000. (1) = $1000.
(1) = double character size.
Must be reset (0).
NMI generation enable (1).

| | | |
|---|---|---|
| $2001 | PCR2 | PPU control register 2. |



Color mode selection. (0) = color. (1) = monochrome.
Blank leftmost column of background display.
Blank leftmost column of sprites displayed.
Blank entire background.
Blank all sprites. (Bits 3,4, when reset, allow CPU access to PPU memory)
Control of background color D0
Control of background color D1
Control of background color D2

| | | |
|---|---|---|
| $2002 | PSTR | PPU status register<br>Read only register. Bit 7 set when interrupt has occurred. Reading this register clears bit 7. |
| $2003 | PPCR | DMA page count register. |
| $2004 | PPAR | DMA page address register. |
| $2005 | PSOR | PPU Scrolling Register. Used to write scroll offset to PPU. Accessed by a 16 bit write, horizontal data first. |
| $2006 | PMAR | DMA Address Register (16 bits, MSB first). Before communicating with PPU memory the starting PPU address has to be loaded into this register. |
| $2007 | PMDR | DMA Data Register (bidirectional). After setting up PMAR successive read/writes can be performed with PPU memory. Note that the first byte read after setting up PMAR will always contain invalid data and should be ignored. |

Background colors.

| | |
|---|---|
| $3F00 | color set #1 background (not used) |
| $3F01 | color set #1 color for bit combo 01 |
| $3F02 | color set #1 color for bit combo 10 |
| $3F03 | color set #1 color for bit combo 11 |
| $3F04 | color set #2 background (not used) |
| $3F05 | color set #2 color for bit combo 01 |
| $3F06 | color set #2 color for bit combo 10 |
| $3F07 | color set #2 color for bit combo 11 |
| $3F08 | color set #3 background (not used) |
| $3F09 | color set #3 color for bit combo 01 |
| $3F0A | color set #3 color for bit combo 10 |
| $3F0B | color set #3 color for bit combo 11 |
| $3F0C | color set #4 background (not used) |
| $3F0D | color set #4 color for bit combo 01 |
| $3F0E | color set #4 color for bit combo 10 |
| $3F0F | color set #4 color for bit combo 11 |

Sprite Colors.

| | |
|---|---|
| $3F10 | Screen background color. |
| $3F11 | color set #1 color for bit combo 01 |
| $3F12 | color set #1 color for bit combo 10 |
| $3F13 | color set #1 color for bit combo 11 |
| $3F14 | color set #2 background (not used) |
| $3F15 | color set #2 color for bit combo 01 |
| $3F16 | color set #2 color for bit combo 10 |
| $3F17 | color set #2 color for bit combo 11 |
| $3F18 | color set #3 background (not used) |
| $3F19 | color set #3 color for bit combo 01 |
| $3F1A | color set #3 color for bit combo 10 |
| $3F1B | color set #3 color for bit combo 11 |
| $3F1C | color set #4 background (not used) |
| $3F1D | color set #4 color for bit combo 01 |
| $3F1E | color set #4 color for bit combo 10 |
| $3F1F | color set #4 color for bit combo 11 |

Sound Voice 1.

| | | |
|---|---|---|
| $4000 | V1R1 | Voice 1 Duty cycle, amplitude and sound length. |
| $4001 | V1R2 | Voice 1 frequency sweep |
| $4002 | V1R3 | Voice 1 frequency lower 8 bits. |
| $4003 | V1R4 | Voice 1 frequency upper 3 bits and actual sound length. |

Sound Voice 2.

| | | |
|---|---|---|
| $4004 | V2R1 | Voice 2 Duty cycle, amplitude and sound length. |
| $4005 | V2R2 | Voice 2 frequency sweep |
| $4006 | V2R3 | Voice 2 frequency lower 8 bits. |
| $4007 | V2R4 | Voice 2 frequency upper 3 bits and actual sound length. |

Sound Voice 3.

| | | |
|---|---|---|
| $4008 | V3R1 | Voice 3 sound length. |
| $4009 | V3R2 | Not Used. |
| $400A | V3R3 | Voice 3 frequency lower 8 bits. |
| $400B | V3R4 | Voice 3 frequency upper 3 bits and actual sound length. |

Sound Voice 4.

| | | |
|---|---|---|
| $400C | V4R1 | Voice 4 amplitude and sound length. |
| $400D | V4R2 | Not Used. |
| $400E | V4R3 | Voice 4 frequency range lower 4 bits. |
| $400F | V4R4 | Voice 4 actual sound length. |

Sound Voice 5.

| | | |
|---|---|---|
| $4010 | V5R1 | Sample clock, repeat and IRQ enable. |
| $4011 | V5R2 | Direct D/A data. |
| $4012 | V5R3 | Delta modulation address pointer. |
| $4013 | V5R4 | Delta modulation byte count. |
| $4014 | DPNR | DMA source page #, high byte used for sprite parameter DMA transfer. |
| $4015 | VMER | Voice Master Enable Register |
| $4016 | JOY1PRT | Controller 1 data latch signal. Contains serial controller data after latch. |
| $4017 | JOY2PRT | Controller 2 data latch signal. Contains serial controller data after latch. |

| | |
|---|---|
| $6000 - $7FFF | Extra CPU Memory (option on cart) |
| $8000 - $BFFF | Program Memory Lower 16K (On Cart) |
| $C000 - $FFFF | Program Memory Upper 16K (On Cart) |

## NES CARTRIDGE EMULATOR

The Nintendo Cartridge Emulator (NCE) was designed by the author. It is the first pass at creating a development system for the NES. The board consists of two dual ported areas of RAM which are accessible by the host as well as the NCE. Development software currently exists for an IBM AT host. The software includes a 6502 cross assembler (XASM), a 6502 monitor (XMON), a background character editor (NICHED), a sprite editor and animator (NISPED), a music and sound effects compiler (NIMCO) and various utilities. The documentation for most of these programs is separate.

The utilities include a downloader and a memory tester. For purposes of adapting your own development environment to an NCE board a listing of the 8088 download utility source code is included. Since the NCE connects to the host through a standard Parallel Printer Port other system may be utilized by following the code principles from the included source file.

## NCE COMMUNICATIONS CODE

Following is the 8088 code used to communicate with the NCE.

```
;*==========================================================*
;*=                  NCE Download Routines                 =*
;*=            Written 1/23/88 by A. Haroutunian           =*
;*=                                                        =*
;*= These routines can be included with TURBO PASCAL       =*
;*= Version 4.0 or above programs that communicate with    =*
;*= the NCE-1 development board. The printer port used     =*
;*= should be specified by the master program. To perform  =*
;*= the read function the printer port must be modified    =*
;*= for bidirectional communication.                       =*
;*= For specific software interface requirements refer to  =*
;*= your TURBO PASCAL manuals.                             =*
;*==========================================================*

DATA            SEGMENT WORD PUBLIC

                EXTRN   DataPort :WORD
                EXTRN   CtrlPort :WORD
                EXTRN   CtrlA    :BYTE
                EXTRN   CtrlC    :BYTE

DATA            ENDS

CODE            SEGMENT BYTE PUBLIC

                PUBLIC  WritePRG
                PUBLIC  WriteCHR
                PUBLIC  ReadPRG
                PUBLIC  ReadCHR
                PUBLIC  NCEInit

                PUBLIC  SetPHost
                PUBLIC  SetPTrg1
                PUBLIC  SetPTrg2
                PUBLIC  SetCHost
                PUBLIC  SetCTrg1
                PUBLIC  SetCTrg2

                ASSUME  CS :CODE

CWR0            EQU     0AH         ; write data to PPI port A
CWR1            EQU     0EH         ; write data to PPI port B
CWR2            EQU     02H         ; write data to PPI port C
```

```
CWR3          EQU     06H          ; write data to PPI control
port

CRD1          EQU     2DH          ; read data from port B

CSEL0         EQU     08H          ; select port A
CSEL1         EQU     0CH          ; select port B
CSEL2         EQU     00H          ; select port C
CSEL3         EQU     04H          ; select control port

MDAOBOCO      EQU     80H          ; all PPI ports outputs
MDAOBICO      EQU     82H          ; B input

RDPULSE       EQU     01H          ; read pulse
WRPULSE       EQU     02H          ; write pulse
LOPULSE       EQU     04H          ; latch low pulse
HIPULSE       EQU     08H          ; latch high pulse

              ;
              ; Write AL to PPI Port A
              ;

PortAWrite    PROC    NEAR
              MOV     DX,DataPort      ; printer data port
              OUT     DX,AL            ; write data
              ADD     DL,2             ; control port
              MOV     AL,CSEL0         ; reg 0 select
              MOV     AH,CWR0          ; port 0 write
              OUT     DX,AL            ; do select
              *XCHG    AL,AH            ; get write
              OUT     DX,AL            ; do write
              *XCHG    AL,AH            ; get select
              OUT     DX,AL            ; do select
              RET
PortAWrite    ENDP

              ;
              ; Update port A with new value
              ; BH is value mask, BL is new bit
              ;

PortAUpd      PROC    NEAR
              MOV     AL,CtrlA         ; get current port value
              AND     AL,BH            ; mask out specific bit
              OR      AL,BL            ; set specific bit
              MOV     CtrlA,AL         ; save new value
              JMP     PortAWrite       ; write it
PortAUpd      ENDP
```

```
                    ;
                    ; Write AL to PPI Port B
                    ;

PortBWrite          PROC    NEAR
                    MOV     DX,DataPort         ; printer data port
                    OUT     DX,AL               ; write data
                    ADD     DL,2                ; control port
                    MOV     AL,CSEL1            ; reg 1 select
                    MOV     AH,CWR1             ; port 1 write
                    OUT     DX,AL               ; do select
                    *XCHG    AL,AH               ; get write
                    OUT     DX,AL               ; do write
                    *XCHG    AL,AH               ; get select
                    OUT     DX,AL               ; do select
                    RET
PortBWrite          ENDP

                    ;
                    ; Read PPI port B
                    ;

PortBRead           PROC    NEAR
                    MOV     DX,DataPort         ; printer data port
                    ADD     DL,2                ; control port
                    MOV     AL,CSEL1            ; reg 1 select
                    OUT     DX,AL               ; do select
                    MOV     AL,CRD1             ; read reg 1
                    OUT     DX,AL               ; do read
                    SUB     DL,2                ; printer port
                    IN      AL,DX               ; read data
                    ADD     DL,2                ; control port
                    PUSH    AX                  ; save input
                    MOV     AL,CSEL1            ; reg 1 select
                    OUT     DX,AL               ; do select
                    POP     AX                  ; restore input
                    RET
PortBRead           ENDP

                    ;
                    ; Write AL to PPI Port C
                    ;

PortCWrite          PROC    NEAR
                    MOV     DX,DataPort         ; printer data port
                    OUT     DX,AL               ; write data
                    ADD     DL,2                ; control port
```

```
                MOV     AL,CSEL2          ; reg 2 select
                MOV     AH,CWR2           ; reg 2 write
                OUT     DX,AL             ; do select
                *XCHG    AL,AH             ; get write
                OUT     DX,AL             ; do write
                *XCHG    AL,AH             ; get select
                OUT     DX,AL             ; do select
                RET
PortCWrite      ENDP

                ;
                ; Update port C with new value
                ; BH is value mask, BL is new bit
                ;

PortCUpd        PROC    NEAR
                MOV     AL,CtrlC          ; get current port value
                AND     AL,BH             ; mask out specific bit
                OR      AL,BL             ; set specific bit
                MOV     CtrlC,AL          ; save new value
                JMP     PortCWrite        ; write it
PortCUpd        ENDP

                ;
                ; Write AL to PPI Control Port
                ;

PortXWrite      PROC    NEAR
                MOV     DX,DataPort       ; printer data port
                OUT     DX,AL             ; write data
                ADD     DL,2              ; control port
                MOV     AL,CSEL3          ; reg 3 select
                MOV     AH,CWR3           ; reg 3 write
                OUT     DX,AL             ; do select
                *XCHG    AL,AH             ; get write
                OUT     DX,AL             ; do write
                *XCHG    AL,AH             ; get select
                OUT     DX,AL             ; do select
                RET
PortXWrite      ENDP

                ;
                ; Select program RAM for download
                ;

SelectPRG       PROC    NEAR
                MOV     BX,7F00H          ; AND/OR Mask
                JMP     PortAUpd          ; 0xxxxxxx
```

```
SelectPRG    ENDP

             ;
             ; Select character RAM for download
             ;

SelectCHR    PROC   NEAR
             MOV    BX,7F80H        ; AND/OR Mask
             JMP    PortAUpd        ; 1xxxxxxx
SelectCHR    ENDP

             ;
             ; Select program RAM host mode
             ;

SetPHost     PROC   NEAR
             MOV    BX,0FC00H       ; AND/OR Mask
             JMP    PortCUpd        ; xxxxxx00
SetPHost     ENDP

             ;
             ; Select program RAM Type 1 mode
             ;

SetPTrg1     PROC   NEAR
             MOV    BX,0FC01H       ; AND/OR Mask
             JMP    PortCUpd        ; xxxxxx01
SetPTrg1     ENDP

             ;
             ; Select program RAM Type 2 mode
             ;

SetPTrg2     PROC   NEAR
             MOV    BX,0FC02H       ; AND/OR Mask
             JMP    PortCUpd        ; xxxxxx10
SetPTrg2     ENDP

             ;
             ; Select character RAM host mode
             ;

SetCHost     PROC   NEAR
             MOV    BX,0F300H       ; AND/OR Mask
             JMP    PortCUpd        ; xxxx00xx
SetCHost     ENDP

             ;
```

```
                ; Select character RAM Type 1 mode
                ;

SetCTrg1        PROC    NEAR
                MOV     BX,0F304H         ; AND/OR Mask
                JMP     PortCUpd          ; xxxx01xx
SetCTrg1        ENDP

                ; Select character RAM Type 2 mode

SetCTrg2        PROC    NEAR
                MOV     BX,0F308H         ; AND/OR Mask
                JMP     PortCUpd          ; xxxx10xx
SetCTrg2        ENDP

                ;
                ; Set address bit 16 for program RAM
                ;

SetAd16         PROC    NEAR
                MOV     BX,07F80H         ; AND/OR Mask
                JMP     PortCUpd          ; 1xxxxxxx
SetAd16         ENDP

                ;
                ; Reset address bit 16 for program RAM
                ;

ResAd16         PROC    NEAR
                MOV     BX,07F00H         ; AND/OR Mask
                JMP     PortCUpd          ; 0xxxxxxx
ResAd16         ENDP

                ;
                ; Set PPI ports A/B/C to output mode
                ;

SetMode1        PROC    NEAR
                MOV     AL,MDAOBOCO
                JMP     PortXWrite
SetMode1        ENDP

                ;
                ; Set PPI Ports A/C to ouput, B to input
                ;

SetMode2        PROC    NEAR
                MOV     AL,MDAOBICO
                JMP     PortXWrite
SetMode2        ENDP

                ;
                ; Generate a pulse on port A
                ; pulse the bit in BL
                ;

DoPulse         PROC    NEAR
                MOV     AL,CtrlA          ; get current value
                OR      AL,BL             ; include new bit
                CALL    PortAWrite        ; set new bit
                MOV     AL,CtrlA          ; restore control value
                CALL    PortAWrite        ; write out
                RET
DoPulse         ENDP

                ;
                ; Send download address to latches 1 and 2
                ; download address in AX
                ;

SendAddr        PROC    NEAR
                PUSH    AX                ; save address
                CALL    PortBWrite        ; write LSB
                MOV     BL,LOPULSE        ; select LOW pulse
                CALL    DoPulse           ; do pulse
                POP     AX                ; restore address
                *XCHG    AL,AH             ; get MSB
                CALL    PortBWrite        ; write MSB
                MOV     BL,HIPULSE        ; select HIGH pulse
                CALL    DoPulse           ; do pulse
                RET
SendAddr        ENDP

                ;
                ; Initialize the NCE
                ;

NCEInit         PROC    NEAR
                CALL    SetMode1          ; all ports outputs
                XOR     AL,AL             ; clear control values
                MOV     CtrlA,AL
                MOV     CtrlC,AL
                RET
NCEInit         ENDP
```

```
                 ;
                 ; Write a block of memory to
                 ; program RAM in the NCE.
                 ;
                 ; Header is:
                 ; PROCEDURE WritePRG(WriteTo    :LONGINT;
                 ;                    WriteFrom  :POINTER;
                 ;                    HowMany    :WORD);
                 ;

WrPCount         EQU    WORD  PTR [BP+4]    ; number of bytes
WrPFrom          EQU    DWORD PTR [BP+6]    ; source buffer adr
WrPToLo          EQU    WORD  PTR [BP+10]   ; dest address lo
wrPToHi          EQU    WORD  PTR [BP+12]   ; dest address hi

WritePRG         PROC   NEAR
                 PUSH   BP
                 MOV    BP,SP
                 CALL   SelectPRG         ; select PRG RAM
                 CALL   SetPHost          ; select host mode
                 MOV    SI,WrPCount       ; get number of bytes
                 LES    DI,WrPFrom        ; ES:DI = source pointer
                 MOV    AX,WrPToHi        ; get high bit
                 CMP    AX,0              ; see if set
                 JNE    Wp1               ; no
                 CALL   ResAd16           ; reset address 16
                 JMP    Wp2               ; continue
Wp1:             CALL   SetAd16           ; set address 16
Wp2:             MOV    AX,WrPToLo        ; get target address lo
                 CALL   SendAddr          ; set up address
                 INC    WrPToLo           ; bump destination adr
                 MOV    AL,ES:[DI]        ; get byte to write
                 INC    DI                ; bump source pointer
                 CALL   PortBWrite        ; write data
                 MOV    BL,WRPulse        ; get write pulse
                 CALL   DoPulse           ; do the write
                 DEC    SI                ; decrement byte count
                 JNE    Wp2               ; continue if more

                 POP    BP                ; restore base
                 RET    10                ; adjust stack
WritePRG         ENDP

                 ;
                 ; Read a block of memory from
                 ; program RAM in the NCE.
                 ;
                 ; Header is:
```

```
                ; PROCEDURE ReadPRG(ReadFrom :LONGINT;
                ;                   ReadTo   :POINTER;
                ;                   HowMany  :WORD);
                ;

RdPCount        EQU   WORD  PTR [BP+4]   ; number of bytes
RdPTo           EQU   DWORD PTR [BP+6]   ; destination address
RdPFromLo       EQU   WORD  PTR [BP+10]  ; source address lo
RdPFromHi       EQU   WORD  PTR [BP+12]  ; source address hi

ReadPRG         PROC  NEAR
                PUSH  BP
                MOV   BP,SP
                CALL  SetPHost         ; select host mode
                CALL  SelectPRG        ; select program RAM
                MOV   SI,RdPCount      ; get number of bytes
                LES   DI,RdPTo         ; ES:DI = dest pointer
                MOV   CX,RdPFromHi     ; get source hi
Rp1:            MOV   AX,RdPFromLo     ; get source lo
                CALL  SendAddr         ; set up address
                INC   RdPFromLo        ; bump source address
                CALL  SetMode2         ; PPI port B = input
                CMP   CL,0             ; see if upper set
                JE    Rp2              ; no
                CALL  SetAD16          ; set upper
Rp2:            MOV   AL,CtrlA         ; get current value
                OR    AL,RDPulse       ; get read pulse
                CALL  PortAWrite       ; make printer port input
                CALL  PortBRead        ; read a byte
                STOSB                  ; save it
                CALL  SetMode1         ; make port B output
                DEC   SI               ; decrement byte count
                JNE   Rp1              ; continue if more
                POP   BP               ; restore base
                RET   10               ; adjust stack
ReadPRG         ENDP

                ;
                ; Write a block of memory to
                ; character RAM in the NCE.
                ;
                ; Header is:
                ; PROCEDURE WriteCHR(WriteTo   :WORD;
                ;                    WriteFrom :POINTER;
                ;                    HowMany   :WORD);
                ;

WrCCount        EQU   WORD  PTR [BP+4]   ; number of bytes
```

```
WrCFrom      EQU    DWORD PTR [BP+6]    ; source address
WrCTo        EQU    WORD  PTR [BP+10]   ; destination address

WriteCHR     PROC   NEAR
             PUSH   BP
             MOV    BP,SP
             CALL   SelectCHR          ; select CHR RAM
             CALL   SetCHost           ; select host mode
             MOV    SI,WrCCount        ; SI = number of bytes
             LES    DI,WrCFrom         ; ES:DI = source
Wc1:         MOV    AX,WrCTo           ; destination address
             CALL   SendAddr           ; write it out
             INC    WrCTo              ; bump destination
             MOV    AL,ES:[DI]         ; get a source byte
             INC    DI                 ; bump source pointer
             CALL   PortBWrite         ; write it
             MOV    BL,WRPulse         ; get write pulse
             CALL   DoPulse            ; do the write
             DEC    SI                 ; decrement byte count
             JNE    Wc1                ; continue if more
             POP    BP                 ; restore base
             RET    8                  ; adjust stack
WriteCHR     ENDP

             ;
             ; Read a block of memory from
             ; character RAM in the NCE.
             ;
             ; Header is:
             ; PROCEDURE ReadCHR(ReadFrom   :WORD;
             ;                   WriteTo    :POINTER;
             ;                   HowMany    :WORD);
             ;

RdCCount     EQU    WORD  PTR [BP+4]    ; number of bytes
RdCTo        EQU    DWORD PTR [BP+6]    ; source pointer
RdCFrom      EQU    WORD  PTR [BP+10]   ; destination pointer

ReadCHR      PROC   NEAR
             PUSH   BP
             MOV    BP,SP
             CALL   SelectCHR          ; select CHR RAM
             CALL   SetCHost           ; select host mode
             MOV    SI,RdCCount        ; number of bytes to read
             LES    DI,RdCTo           ; ES:DI = destination
pointer
Rc1:         MOV    AX,RdCFrom         ; get source address
             CALL   SendAddr           ; write it out
```

```
                    INC     RdCFrom          ; bump source address
                    CALL    SetMode2         ; PPI port B is input
                    CALL    SelectCHR        ; restore char select
                    MOV     AL,CtrlA         ; get control value
                    OR      AL,RDPulse       ; get read command
                    CALL    PortAWrite       ; make printer port input
                    CALL    PortBRead        ; read the byte
                    STOSB                    ; save it
                    CALL    SetMode1         ; make port B output
                    CALL    SelectCHR        ; restore char select
                    DEC     SI               ; decrement byte count
                    JNE     Rc1              ; continue if more
                    POP     BP               ; restore base
                    RET     8                ; adjust stack
ReadCHR             ENDP

CODE                ENDS

                    END
```